MASTER

Coding policies for secure web applications

Samuel, S.

Award date:
2007

TECHNISCHE UNIVERSITEIT EINDHOVEN
Department of Mathematics and Computer Science

Coding Policies for Secure Web Applications

By
Sabrina Samuel

Supervisors
Dr ir L.A.M. (Berry) Schoenmakers (TU/e)
Drs S.B. (Sander) Reerink CISSP (PricewaterhouseCoopers Advisory N.V.)

Eindhoven, November 2007

ACKNOWLEDGEMENTS

I take this opportunity to thank all those who have in one way or another help me see through the completion of this Master thesis.

I thank Berry Schoenmakers, my supervisor, for lending me his guidance throughout the writing of this thesis. Berry shared with me a lot of his ideas that helped me improve the structure, content and value of this thesis.

To my supervisor at PricewaterhouseCoopers, Sander Reerink, thank you for all your technical advises and support. The many discussions we had helped me to see things from different perspectives especially in the technical aspects of this thesis.

I wish to extend my sincere gratitude to Benne de Weger for accepting to be in the evaluation committee of my final examination.

I express my deepest appreciation to Jim Portegies for his support and help throughout the writing of this thesis.

I would also like to thank all of my good friends, most importantly Chibuzo Obi, Olivier Toelen, Kasturi Dewi, and Alejendro Mendoza, for sparing their time in reading and providing feedback on various parts of my thesis. Not forgetting all of my other good friends, both in Malaysia and in Eindhoven, thank you so much for your support, motivation and care.

Last but not least, to the important people in my life, Sabbastian Raj, Hilda Fernandez (my Mother), Samuel Arokiaraj (my Father) and the rest of my family; you saw me through the entire Masters program and you have given me immense support, encouragement and confidence that I can finish it. I thank you for you constant prayers.

ABSTRACT

The increased volume of transaction and communication over the World Wide Web in industries like banking, insurance, healthcare, travel and many others has triggered a number of unprecedented security issues. Most web applications today are susceptible to attacks ranging from unauthorized access, movement, alteration or deletion of files, virus attacks, and thefts of data. The use of perimeter defenses like firewalls, anti-viruses and the likes are insufficient. Because of this, industries are seeking for more comprehensive security measures that can be incorporated in their web applications. An inclusion of defense which will evidently reduce vulnerabilities in web applications is seen to be in the development lifecycle of the application itself. Developers need to learn and examine the vulnerabilities that could possibly occur in web applications so that precautionary measures can be adopted in the implementation stage. This thesis serves as an elementary guideline for all those involved in the application's development process and more importantly designs and formulates a set of secure coding policies and guidelines as pro-active remediation strategies to strengthen the security of web applications.

CONTENTS

Introduction

In recent times, the reliance on information and services offered through the web has increased the expectations at all levels of web applications usage, from the casual surfers through to large business corporations whose business strategies are underpinned by secure and reliable web services. This in turn has generated more awareness on the fundamental information security best practices that should be achieved by every web application. These include confidentiality, integrity and availability. For this reason, it has become imperative for the affected industries to take precautionary measures to prevent breaches in information security by establishing an efficient development framework which would be able to withstand the dynamics of web applications security without compromising its operational dependability.

The increasing use of web applications and the growing number of exploits is one of the primary motivations for gathering, explaining and analyzing the details of web application security. Studies reveal that while there are plentiful resources including articles, conferences, and organizations that are dedicated to educating people on the importance of information security, almost none of the resources found were anywhere near to being as comprehensive as necessary for web developers. Most programming books or tutorials fail to address security issues and most security resources miss the essential programming details for secure coding. Bearing this in mind, the major part of this thesis is targeted towards formulating a set of coding policies and guidelines that will act as a checklist to assist the web application development team in coding securely.

Due to the long-term gain in cost, time efficiency and reputation, many organizations are beginning to emphasize the importance of embedding security controls in their business applications, specifically during the applications design and development stage. Hence, a part of this thesis outlines how security initiatives should be adopted at each stage of the application development lifecycle. This thesis serves as an elementary guideline for all those involved in the application's development process i.e. requirements engineers, architects, designers, developers and testers.

This thesis consists primarily of four (4) chapters.

Chapter 1:	Studies and elaborates how security related activities are to be included in each phase of a web application development lifecycle.
Chapter 2:	Analyses and examines the strategies used by attackers to take advantage of the vulnerabilities that exist in web applications to compromise security.
Chapter 3:	Informs the readers of various existing libraries, classes, frameworks and related components in two of the most prominent web development languages used today.
Chapter 4:	Provides a checklist of coding policies and guidelines that will assist developers in designing and developing secure applications.

In the first chapter, the reader is given a high-level perspective of how security controls fit into the application development lifecycle. Security must be made an integral part of every application's development lifecycle. Chapter 2 discusses the ten foremost web application security threats and vulnerabilities affecting today's web applications. The reader is walked through some real world examples of web application attacks. At the end of chapter 2, the reader is expected to have an idea of the important measures that must be in place to avoid the vulnerabilities. Chapter 3 can be seen as a supporting chapter for experienced developers to obtain knowledge of the various existing libraries, classes, and frameworks of the two widely used web development languages namely Java and .NET, to curb the web application vulnerabilities discussed. Finally, Chapter 4 which is considered to be the main part of this thesis; establishes a set of coding policies and guidelines, based on the extensive study and analysis in previous chapters. It is expected to be a valuable checklist for web developers.

Additional information including a brief description of the two primary technologies discussed in the thesis can be found in the appendices. A quick reference card is also designed to give adequate information for developers who seek to obtain information fast. The reference card is hoped to give developers using either the Java or .NET environment, compact coding policies to circumvent the discussed web application vulnerabilities.

1 Security in the Software Development Lifecycle (SDLC)

The increasing use of the web to access information and request services has led many organizations, irrespective of their business activity, to incorporate web development as part of their business. The web as we know today is not only used to advertise information or enable services and products to be purchased, but has grown to incorporate more and more flexibility as well as interactive functionality. Some examples of web applications include e-commerce/e-business web sites, search engines, transaction engines and informational web sites conveying news, advertisements, articles and many others. Advancements in communication technologies and web enabled appliances further explain the evolvement of web applications being used today. In the future, the use of the web is perceived to grow exponentially with a variety of added services in most business sectors [8, 9, 10].

The growing dependency on the range of web applications necessitates the development of secure and reliable web applications. Hence, organizations are seeking for a more comprehensive development lifecycle that will aid in reducing security breaches. For a long time, a lot of attention was only given to strengthen the security of networks. This led attackers to shift attacking strategies from the network layer to the application layer. Besides the Internet evolution, the lack of awareness in application vulnerabilities has caused the rise of attacks on the application layer [15, 11]. Evidently, according to SPI Dynamics, Inc and the *Internet Security Threat Report* from Symantec, more than 70 percent of all hacking events of today occur at the application level. For this reason, organizations are striving to incorporate sufficient measures into an application's development lifecycle to make sure that eventually both the application and the network are deemed secure under malicious attack attempts.

A development lifecycle entailing secure web applications is similar to the general development lifecycle for system applications except with the inclusion of adequate security analysis, defences and countermeasures. There exist many lifecycle models, each defining specific methods of execution in an application's lifecycle. Famous examples of application development lifecycle models are the iterative, agile and waterfall model. In most application development, as detailed in [3], it is important that an organization first understands the processes it must adopt to build secure applications. If the processes are not well understood, it will be hard to determine its weaknesses and strengths which will consequently impede the continuous improvement of the process. Furthermore, by using a common framework, an organization can set its own standards and security goals to achieve its intended web application.

A typical and complete application development lifecycle, consisting of 7 stages is as depicted in Figure 1. Slightly varying from McGraws version in [6], Figure 1 illustrates the assimilation of security into all stages of an application lifecycle. Essentially, a secure application development process is primarily intended for application developers and software architects. However, practically, as also mentioned in [1], security must be thought and practiced by all those who are involved in the development lifecycle. This includes the requirements engineers, architects, designers, developers, testers, and users. A misstep in any one of the stages can cause severe impact to the end product. Gartner Research [2] realized that the cost of addressing security vulnerabilities during the development cycle is less than two percent the cost of removing a defect from a deployed production application. Moreover, in [7], Gartner reports that applications without sufficient protection at the application layer will eventually face extinction. Yet another appalling prediction by Gartner is that by the year 2009, 80% of enterprises will fall victim to an application layer attack.

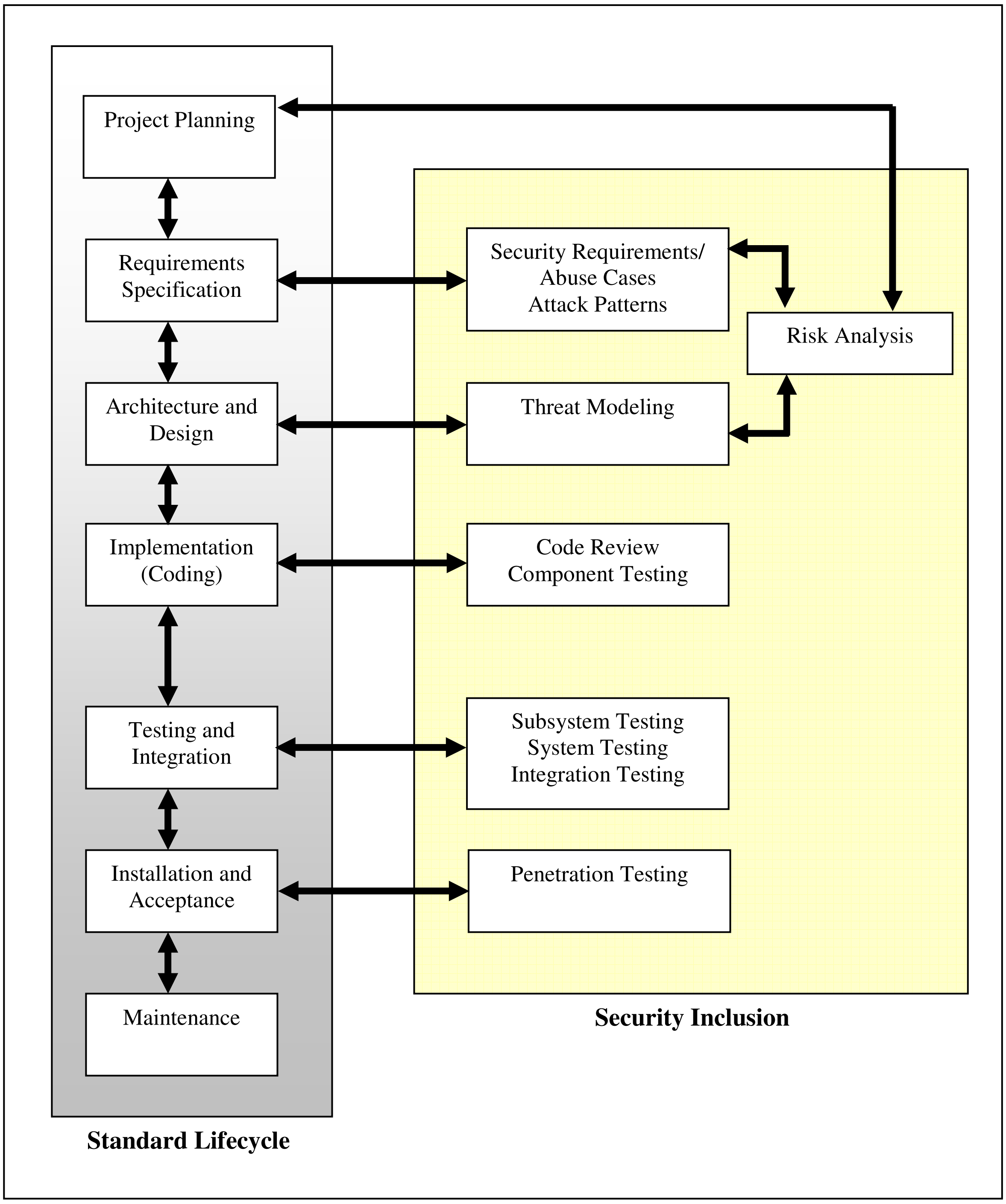

Project Planning
Requirements Specification
Architecture and Design
Implementation (Coding)
Testing and Integration
Installation and Acceptance
Maintenance
Standard Lifecycle
Security Requirements/ Abuse Cases Attack Patterns
Risk Analysis
Threat Modeling
Code Review Component Testing
Subsystem Testing System Testing Integration Testing
Penetration Testing
Security Inclusion

Figure 1: Secure Application Development Lifecycle

Section 1.1 describes the activities of each stage, using the iterative model approach of Figure 1, and how security initiatives can be applied to establish a secure application development lifecycle. In general, the iterative approach is best used as each stage will be revisited more than once as the application evolves. For better understanding, a suitable example, a web based Internet banking application is used to show how the various lifecycle stages can be executed.

An Internet banking application enables users having accounts in particular banks to access and manage accounts and contracts like loans, mortgages, and insurance. The application facilitates transactions such as online transfer and payments, cheque issuance, investments in bonds and equity, and various other banking services. All of these services should be accessible at all times unless specific notices are given due to updates and/or maintenance.

1.1 Stages in the SDLC

1.1.1 Project Planning

The first stage in the SDLC is the planning stage. Needless to say, the planning stage is indispensable. It is needed to obtain an overall conception of the intended application in order to establish the development schedule and timeline, evaluate the feasibility and risks associated with the application as well as to decide on appropriate management and technical approaches in implementing the application.

By first understanding the business context, the application's business goals can be clearly derived. This may be done using the Goal Question Metric paradigm framework [56]. At this stage the project planners identify the groups and individuals responsible for accomplishing various assigned tasks, the time and resources required as well as data collection, analysis and reporting procedures. Subsequently, a reasonable estimate of development schedule and timeline can be agreed. This estimate however is continually refined and improved as the work progresses. Because of this, the plan should also include a framework for negotiating time and resources in case there would be a delay.

Next, the task of defining and ranking priorities and circumstances with respect to the kind of security risks associated with individual business goal is outlined. Educating business personnel on the exact nature of security controls, and how they affect timelines, budgets and where they fit into the overall process is an important part of early stakeholder engagement process [60]. The business attempts to identify the possibly successful exploits based on the environment to which the application would be operating; the potential business impact resulting from the exploits, the technologies used and what mitigation steps can be taken to manage and control the damage. This results in a preliminary feasibility and risk assessment process which captures the basic security requirements to which the application should satisfy. This also helps to clarify and quantify the direct impact (Low/Medium/High) of certain events such as unexpected system crashes, unauthorized data modification or disclosure, will have on the business goals.

Using the information obtained from the feasibility and risk assessment process, the organization is able to estimate the likely costs of liability, redevelopment and reputation damage. Moreover, with a combined management and technical implementation decisions, a more realistic schedule incorporating security aspects particularly in the application's development stage will be forethought. This would certainly help in establishing a more accurate development cost, schedule and timeline.

From a security viewpoint, the identification of risks which precedes the risk mitigation strategy at an elementary level is valuable at this stage because it provides requisite confidence in addressing security concerns that could arise during the lifetime of the application. The risks and mitigation strategy is later reassessed as the application progresses through the subsequent stages, particularly in the architecture and design stage. Completing this exercise enables an organization to include adequate security analysis early in a development lifecycle which would result in a less expensive and more effectively secure application than having to add features and functionality later to an operational system.

An example of a preliminary assessment template is available in Appendix A.

1.1.2 Requirements Specification

After planning comes the requirements specification stage. This stage is one of the most important stages as it attempts to deliver a set of requirements that exhibits the business needs and goals of the intended application. Requirements are normally divided into four categories namely functional requirements, non-functional requirements, constraints and assumptions. Functional requirements refer to the expected behaviour of the application expressed as tasks or functions the application is required to perform. On the other hand, non-functional requirements (which are sometimes referred to as Quality of Service (QoS) requirements), describes the additional properties an application must have. Some examples of additional properties are performance, reliability, portability, usability, etc. Constraints specify the factors that limit the development of certain feature or functionality of the intended application given the conditions for the development. Similarly, all relevant assumptions required for the development of the application must be clearly outlined.

Below are some examples of requirements for an Internet banking application. The requirements written are merely examples and are not in accordance with, for instance the IEEE Recommended Practice for Requirements Specifications. Later, the extensions of these requirements to include security measures are shown.

Functional Requirements:

1. The application shall enable transfer of payment to an account of the same bank or to an account of a different bank/institution/organization.
2. The application shall allow authenticated users to view balances and transaction information on deposits, loans, credit card and mortgage accounts.

Non-Functional Requirements:

1. ***Performance Requirements***
   The response time to sign-in shall be within 5 seconds from the "sign in" submission.
2. ***Security Requirements***
   i. The application shall allow all transactions to be carried out securely in "HTTPS browser retrieval" mode.
   ii. The application shall allow users to retrieve lost or forgotten username and /or password.

Figure 2: Example of requirements for an Internet banking application

Conventionally, as also stated in the ISO 9126 standard, security often falls into the category of non-functional requirements. However, ideally, it is insufficient that security is just considered within the non-functional requirements category especially since most attacks are based on the behaviour of an application which is built mostly from the functional requirements. Therefore, to ensure security as a whole, security concerns should be intertwined together with both the functional and non-functional requirements. In this way, all possible security aspects of an application are given considerable amount of thought and attention for subsequent stages.

Without a clear and complete description of the requirements, it is difficult to proceed to the subsequent stages in the lifecycle. Hence, a close liaison between requirements engineer, architects, designers and developers are vital to ensure that the requirements are complete, correct, consistent and most of all that it can be implemented given the relevant constraints and assumptions.

Security focused requirements are divided into two classes; positive requirements which determine the secure functional behaviour of an application and negative requirements which describes the behaviours an application must avoid [6]. A positive requirement uses the term "shall" or "must" while a negative requirement uses the contrary, i.e. "shall not" or "must not".

Figure 3 shows the reformulated requirements of Figure 2 to supplement the security needs of a web based Internet banking application for both the functional and non-functional requirements.

The negative security requirements class brought forth the concept of abuse (misuse) cases. Abuse cases are used to describe how malicious users might interact with the system. More clearly, when verifying user input for instance, a series of abuse cases can be constructed by investigating and detailing how attackers may possibly attempt to execute attacks that causes buffer overflows, SQL injections, cross site scripting and the like. Abuse cases can also be used to brainstorm methods on how the application can avoid such vulnerabilities and threats. When generating abuse cases, it is valuable to consider similar applications that have been victims of attacks.

Using the Internet banking application, a simple example of use case and abuse case would be as depicted in Figure 4. The abuse case is shaded and it shows the possible interactions between one or more perpetrators (malicious users who can either be an insider or an outsider of the application) that causes harm which affects any part of the system, users involved with the system or the system's stakeholders. The perpetrator is assumed to be tactical enough to gain privilege into the system, perform fraudulent transactions or operations and is able to remove evidence of such actions. For huge complex applications like an Internet banking application, it is good to distinguish the outsider and insider roles of the Perpetrator, including their skills and resources. It is also important to perform careful analysis of the environment to which the system is exposed.

Abuse cases, however, only give an overview of possible attack scenarios. It is vital to also include a textual representation which provides explicit information with respect to the identified cases. A template for textual representation of abuse cases, adapted from [43] is designed. Please refer to Appendix A.

Following the construction of abuse cases, attack patterns can be used as an initial process that helps to develop a more comprehensive set of attack techniques. An attack pattern is defined to be a series of steps to simulate attacks on an application. It helps to identify and qualify the risk that a given exploit will occur [58]. Apart from preventing potential vulnerabilities in the requirements, architecture, design, and eventually in the code, attack patterns are also significantly useful for testing purposes. There are extensive literatures on attack patterns available. In this thesis, information about attack patterns is obtained from [4, 58, 59].

The combined effort from the requirements engineer, architects, designers, developers, testers and users should entail a detailed Software Requirements Specification document (SRS); in this case, software refers to the web application that addresses not only the security requirements but the preferred strategy to control security breaches. Hence, the requirements phase is the most suitable time to determine possible risks so that informed decisions about security tradeoffs can be made. The fundamental key to writing security requirements is to be as specific as possible and to aim at making the requirements testable and measurable [60]. Thoroughly developed SRS containing detailed security requirements can help improve the security of applications and evidently reduce the cost and necessity for re-work.

1.1.3 Architecture and Design

In this architecture and design stage, the architects and designers are responsible for describing and designing elaborately the functionality and features of the proposed application as perceived from the previous stages. An architect develops an abstract representation of the proposed application ensuring that it meets all specified requirements as well as creating room to cater for future requirements or enhancements. Hence, decisions must be made about how the application will be structured, how the requirements are interpreted, how the various components will integrate and interact, and which technologies will be leveraged [6].

Functional Requirements:

1. The application shall enable transfer of payment to an account of the same bank or to an account of a different bank/institution/organization.

   Security inclusion:

   Positive Requirement

   i. The application shall log all transactions to ensure that all cash/cheque transfers are between legitimate accounts.
   ii. All legitimate accounts have been documented in <a-document-name>

   Negative Requirement

   i. The application shall not process more than 2 transactions of more than Euro 3,000.00 within a day.

2. The application shall allow authenticated users to view balances and transaction information on deposits, loans, credit card and mortgage accounts.

   Security inclusion:

   Positive Requirement

   i. The application shall timeout after 15 minutes of inactivity to ensure discontinuity of a previously authentic session.

   Negative Requirement

   i. The application shall not allow users to view balance and transactions of other users.

Non-Functional Requirements:

1. Performance Requirements

The response time to sign-in shall be within 5 seconds from the “sign in” submission.

Security inclusion:

Positive Requirement

i. After 5 seconds has lapsed, the application shall erase all cookies related to the sign-in session.

Negative Requirement

i. The sign- in session cannot be refreshed once the 5 seconds timeline has lapsed.

2. Security Requirements

1. The application shall allow all transactions to be sent securely in “HTTPS browser” mode.

   Security inclusion:

   Positive Requirement

   i. The application shall use the HTTPS/SSL encryption technology once logged in.

   Negative Requirement

   i. The application shall not switch back to the ordinary HTTP mode.

2. The application shall allow users to retrieve lost or forgotten username and /or password.

   Security inclusion:

   Positive Requirement

   During registration:

   i. Each user shall be requested to provide the system with a secret question/hint to which only the user knows the answer.
   ii. Each user shall be requested to provide the system with an email address to which forgotten passwords are sent to.
   iii. Failure to answer the security question in three attempts will result in inaccessibility of the account.

   Negative Requirement

   i. The application shall not send/display forgotten passwords to the provided email account without proper authentication of user identity.
   ii. Locked accounts can only be unlocked manually at the bank.

Figure 3: Inclusion of security measures in an Internet banking application requirements

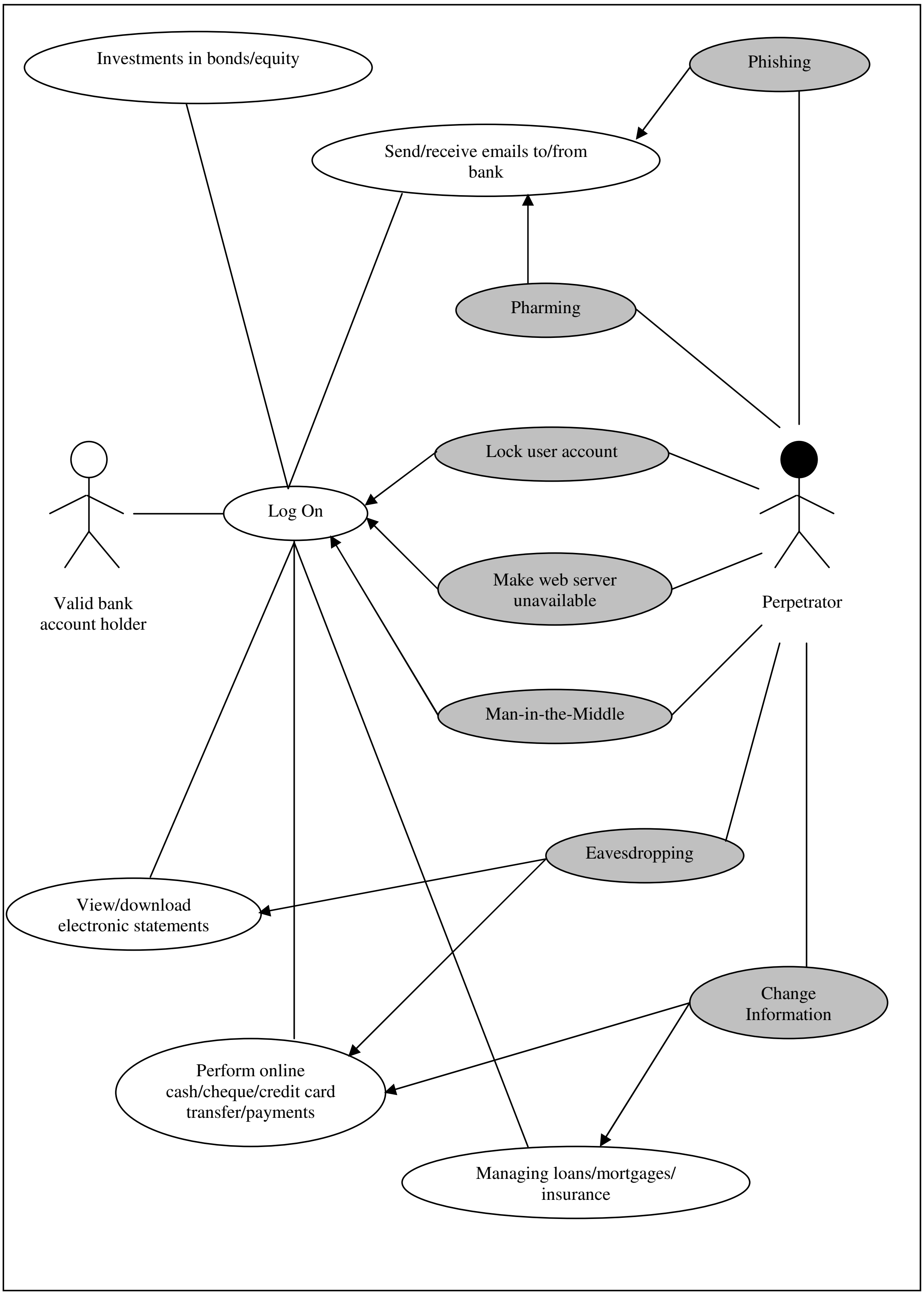

Investments in bonds/equity
Phishing
Send/receive emails to/from bank
Pharming
Lock user account
Log On
Make web server unavailable
Valid bank account holder
Perpetrator
Man-in-the-Middle
Eavesdropping
View/download electronic statements
Change Information
Perform online cash/cheque/credit card transfer/payments
Managing loans/mortgages/ insurance

Figure 4: Use case and Abuse case of a typical Internet banking application

To develop secure web applications, it is essential that architects and designers are well trained and educated for identifying and analyzing possible security risks and vulnerabilities involved based on the developed requirements. Most security vulnerabilities are said to occur more rampantly in the architecture, design and implementation stage. This happens because of poorly designed application architecture, the application underwent rushed implementation and occasionally due to the ignorance of designers and developers.

Architecture must be designed to be resilient against internal and external threats. Undiscovered security gaps at this stage tend to cause cascading impact in the subsequent stages of the application development's lifecycle. Hence, the architecture and design of a web application must ensure adherence to a company's security policy in addition to the risks assessment outlined in the prior stages.

Like the preliminary risk analysis done in the first stage, architects and designers, in co-operation with developers must execute an architectural risk analysis using a method known as *Threat Modelling* to assess security exposures from a technical point of view. In simplicity, threat modelling strives to closely emulate attackers and identify all attack methodologies and their likelihood of success. The abuse case template designed in the previous stage can be used to verify and validate the threat model.

Below are the basic steps required in performing threat modelling [63];

1. Identify protected resources (i.e. customer database)
2. Assign level of criticality to the resources
3. Identify potential attackers, attacks and its likelihood
4. Estimate relative frequency and impacts of such attacks
5. Analyze and determine possible attack routes (i.e. attack tree analysis)
6. Find measures to protect all possible attack routes

While abuse cases, in Stage 2 presents an overview of possible attack scenarios [60], threat models, describe in more detail the attack paths, interfaces and data elements involved in a likely attack. Upon completing both these techniques, architects and designers will have clearer and more organized information about the kind of security needed to implement a secure application.

1.1.4 Implementation

This phase is where the actual coding, based on the design and architecture of the system, is written by a group of developers. A well designed architecture eases the task of developers in writing well-defined components with well-defined interfaces. According to the findings by the Secure Software Forum, while 65% of developers are not confident in their ability to write secure applications, 70% of security problems surface in the application layer. As a result, training and education together with appropriate code auditing or reviewing process as well as close interaction amongst developers, application architects and designers can increase the awareness and practice of secure coding.

Developers should carefully weigh all available options before deciding how to implement each module, taking into account proper error handling mechanisms, avoiding the construction of code that can be compromised, including various encryption techniques as well as ensuring a secure communication platform.

Two main elements that should exist in the implementation stage are *code review* and *component testing*. In code review, a team of developers, testers, architects and designers get together to review the written code and check for correctness, consistency and completeness with respect to the specified requirements, architecture and design. This practice gathers the opinion and mindset of those involved to foresee the vulnerabilities in coding practices as well as to validate the absence of targeted weaknesses. [4] points out that it is important that the basic code reviewing techniques are made more security oriented. For example, thoroughly reviewing input validation modules can reduce the risk of compromising or destroying an entire database. Since the inspection and review process may be impractical for a large application, the use of automated analysis tools available from commercial vendors like Fortify, Klocwork, Coverity, etc can be used and even customized according to organizational needs. Each tool offers a comprehensive and growing rule set depending on the area of focus. The coverage of the accompanying rule sets should be the primary factor when deciding on the right tool from the right vendors [4].

It is also the responsibility of the developers to ensure thorough component testing of each developed function or module. Component testing or unit testing involves the testing of individual functionality of an application as specified in the component design. When these elements have been properly verified and validated, the application is now in position for the Testing and Integration stage.

A closer look on secure coding details, particularly in Java and .NET can be found in Chapters 3 and 4 of this thesis.

1.1.5 Testing and Integration

Here, the application is tested for its overall functionality through various levels of testing including but not limited to subsystem test, system test, system integration test, user acceptance test, release test, etc. Exhaustive testing requires proper planning and should be based primarily on the system's requirements and architecture. In most cases, a suitable test environment must also be well defined to ensure the accuracy and reliability of test results obtained.

There exists a distinction between levels and types of testing. The former is as aforementioned and the latter includes functional testing, interface testing, security testing (i.e. penetration testing), load/stress testing, performance testing, etc. At all levels or types of testing there are three testing methodologies that can be applied. Table 1 provides a short introduction of these methodologies with respect to security [57].

A detailed discussion about the various levels and types of testing, however, is not within the scope of this thesis.

Testing Methodology	**Advantages**	**Disadvantages**
Black Box Testing an application without knowledge on the internal workings (functionality, structure, source code, architecture, etc) of the application.	• Well suited for large complex systems revealing unexpected errors. • Helps to identify the ambiguities and inconsistencies with respect to the functional specifications. • Unbiased testing strategy as both testers and developers are independent of each other.	• The discovery of bugs and/or vulnerabilities can take significantly longer. • Challenging to design test cases, i.e. identify sensitive inputs • Possibilities of missing important execution paths.
White Box Testing an application with knowledge and access to all internal workings of the application.	• Enables a complete testing coverage • Easy to develop test cases as testers have access to existing functions, libraries, and inputs to which the application should receive or reject • Enables code optimization by for instance, removing extra lines of code which may have hidden defects.	• The complexity of architectures and volume of source code introduces challenges. • Increases cost as it requires well experienced and skilful testers • Impossible to look into every piece of code.
Gray Box Testing an application with limited knowledge on the internals of an application. The knowledge is usually	• Offers Combined Benefits from both Black Box and White Box testing • Intelligent test case generation which are based on the	• Partial code coverage • Increased cost in time, skill, repeatability, and overall expense of the testing process.

constrained to the design and/or architecture documents. Also known as Hybrid Analysis.	application's design and architecture documents	

Table 1: Black Box, White Box and Gray Box testing methodologies

From a security viewpoint, this phase contributes largely in identifying the security gaps that exists within an application. It is critical that a complete set of testing procedures is executed not only to look for errors, bugs and interoperability of the functional aspects of an application but also to look for threats and vulnerabilities. Gary McGraw in [6] proposed two strategies for testing, namely testing of security functionality with standard functional testing techniques and risk based security testing based on attack patterns, risk analysis results and abuse cases. Hence, intensive testing of security requirements requires attention to the operating environment of the application such as the operating systems and network connections, to name a few, as well as rigorous functional testing on the implemented security components.

During the testing stage, testers attempt to develop a comprehensive set of test cases stressed more towards discovering potential vulnerabilities in order to verify and validate the security of the application. Selecting appropriate testing approaches is a challenging task as it relies heavily on the project's available timeline, resources as well as the overall objective of the business. Similar to coding, test cases should also undergo review and inspection process. It is also important to understand how testing constraints affect the completeness of testing results. The threat model, attack patterns, abuse cases, risk analysis developed in previous stages can assist testers in understanding the different possible vulnerabilities. This will help testers in envisioning lines of attack in order to develop strategies on how to exploit the vulnerabilities. However, for large complex systems, existing security testing tools like Nessuss, Tripwire, Snort and various others may be used to get a rudimentary analysis of the application.

Evidently, testing can help to

- confirm if developers overlooked some secure coding practices
- find flaws that were not visible during design and development
- provide metrics of an application's security
- measure the effectiveness of risk mitigation activities

1.1.6 Installation and Acceptance

The application at this stage is ready to be installed and deployed in production as per the business goal. Necessary testing activities, namely system penetration test, configuration test, installation test, and acceptance test must be carried out before deployment.

1.1.7 Maintenance

This is the final stage in an application development lifecycle which is a continuous process constantly monitoring, enhancing and/or upgrading the developed application as and when needed. The use of a logging process can assist in monitoring and detecting any misuses or attempted security breaches.

Maintenance is also fundamental to ensure that existing security process and procedures are intact and that all changes/upgrades conforms to particular change management process or incident response plan which are periodically audited for the overall security of the system.

1.2 Summary

In this chapter, we showed that security plays a part in all stages of the application development lifecycle. Beginning from the *planning* stage, we see that educating business personnel with preliminary security assessments assists business to plan for risk mitigations. In the gathering and elicitation of *requirements*, we see that it is insufficient to address security elements under the non-functional requirements category. Security must be intertwined in both functional and non-functional requirements. Important concepts introduced in the *requirements specification* and *architecture and design* stage, like misuse cases, attack patterns, threat modelling seem to be effective in learning the various possible security threats and vulnerabilities in order to strengthen the affected parts of the application. All of these steps deliver a concrete technical specification for the *implementation* and *testing stage*. Additionally, the *implementation* stage involves code reviewing and component testing procedures while the *testing and integration* as well as *installation* and *acceptance* stage includes many different types and levels of testing strategies. The last stage stresses the need for on going process to make sure that future needs and updates do not leave any security gaps.

Secure web applications are only reachable if security gets a position in all stages of the development lifecycle. The study also showed that a close liaison among requirements engineers, architects, designers, developers and testers is necessary as their experience and skill sets are largely complementary. This corporation will more likely yield better security against well known, easily predicted or sometimes foreseeable attacks, guaranteeing users and the business of secure applications which are difficult to exploit.

By strictly considering security focused design and coding principles as an integral part of any web application development lifecycle, it becomes feasible to build and maintain robust, reliable, and trustworthy web applications.

The next chapter takes a closer look at some of the threat and vulnerabilities affecting today’s web applications together with some practical attacking strategies used by attackers.

2 Web Application Security: Threat and Attack Analysis

Continuous advancements in dynamic (interactive) web applications to conduct businesses over the Internet have led to a myriad of threats. These threats raise much apprehension among users of web applications particularly in aspects of retaining confidentiality (privacy) and integrity of personal information. The threats plaguing web applications are manifold and deriving from [12, 15] can be analyzed to manifest principally from three areas; as depicted in Figure 5 below. The highlighted region exhibits the dependencies between the areas.

For detail and thoroughness, the study of threats in web applications is narrowed down to focus solely in the implementation category. However, as seen in Chapter 1, the phases within an application development lifecycle are closely tied with one another. Hence, poorly written source code may be due to an imprecise design decision which in turn is a result of incomplete or missing information from the architecture category.

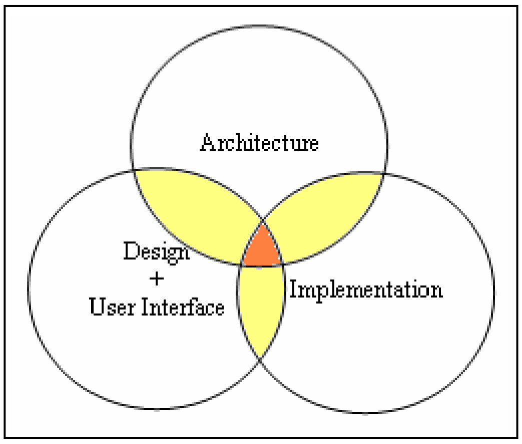


Figure 5: Threat Categories

Category	**Definition**
Architecture	Architecture is derived from a system's requirements document and is concerned with the selection and specification of the prescribed system components and their individual functionalities, the interaction between the system components, as well as the constraints of these components and their interactions.
Design + *User Interface*	Design is concerned with the modularization and detailed interfaces of the components, the specific algorithms and procedures that define the components, and the data types needed to support the architecture and to satisfy the requirements.
Implementation	Implementation refers to the task of developing the actual system. The tasks consist of programming, testing and eventually deploying the newly developed, tested and accepted system.

***Definition extracted from the Software Engineering Institute (SEI), Carnegie Mellon*

The threats as also listed in the Open Web Application Security Project (OWASP) Top Ten (10) Project for web application security is as follows:

1. Unvalidated input
   a. Buffer Overflows
   b. Cross Site Scripting
   c. Injection Flaws
      i. SQL Injections
2. Broken Access Control
3. Broken Authentication and Session Management
4. Improper Error Handling and Logging
5. Insecure Storage
6. Application Denial of Service
7. Insecure Configuration Management

Different from the OWASP Top Ten list, the threats and vulnerabilities here are grouped according to their originating factor. They can also be reordered into three(3) major categories namely *Security Mechanisms* (Broken Access Control, Broken Authentication and Session Management, Insecure Configuration Management, Improper Error Handling and Insecure Storage), *Attack Patterns* (Injection Flaws, Denial of Service) and *Vulnerabilities* (Cross Site Scripting and Buffer Overflows).

Before proceeding, it is worthwhile to understand the difference between the terms vulnerability, threat and attack. The definitions as used in literature are as follows:

- ***Vulnerability***: a weakness of an asset or group of assets that can be exploited by one or more threats [82].

- ***Threat***: Any circumstance or event with the potential to harm an information system (IS) through unauthorized access, destruction, disclosure, modification of data, and/or denial of service, which may result in harm to a system or organization [16, 82].

- ***Attack***: An intentional act of attempting to bypass one or more security controls of an information system (IS) most importantly: confidentiality, integrity, availability, authentication and/or non-repudiation [16].

Therefore, a threat paves open a way for the occurrence of an attack.

The following results the study of each of the above mentioned threats and vulnerabilities and the likely attack scenarios made possible by these threats. In some instances, the correlation between threats becomes obvious where one threat may cause more than one type of attack and vice versa. For example, a successful SQL Injection on a logon functionality is also a breach in access control, a buffer overflow exploit is related to an injection flaw and possibly application denial of service vulnerability.

2.1 Unvalidated Input

In an interactive web application, there are many types of user input. For instance, web applications like Internet banking require its users to logon in order to carry out various banking services. This requires users to first obtain a valid username/ID and password/pin from the bank. Occasionally, certain inputs are retained during the active state of the application for extended processing. For example, after signing-in the username/ID might be stored in a cookie which enables the user to continue using the services from the web site. The manner in which input is captured, retained and/or passed poses a threat as these inputs often directly interface with the application's database/server.

Hence an error in the input can easily cascade into data corruption and/or numerous other security breaches.

User inputs from web applications can be obtained, retained or passed through the URL/query string, headers, cookies, or form fields which include hidden fields.

URL/query string	Input values embedded in the HTTP query string (URL) following the '?' sign and is sent to the server using the GET method.
Headers	Header information sent as part of the HTTP request header also contains information provided by the web server, regarding the current request.
Cookies	Input values associated with a user and is stored on a user's computer for subsequent access to particular web sites.
Form Fields	Inputs obtained from web forms and posted using the POST method.

Table 2: Sources of user input

Apart from input received from the web client, web applications will at some point also receive input from other sources like files and databases.

Today, input validation is seen to be one of the major security measures against web application attacks like buffer overflows, cross site scripting, injection flaws, SQL injections and various others. This is further supported by [17] which reports that 50% to 60% of security vulnerabilities in almost 70% of applications are due to poor input validation. Disappointingly, web application developers often fail to rigorously validate these input fields allowing various types of input to be entered or subverted, hence compromising the security of the application.

There are two types of input validation techniques for web applications; client-side validation and server-side validation. Traditionally, web applications use client-side validation which uses scripting languages like JavaScript or VBScript. This method of validation is performed exclusively at the browser level and helps to ascertain that the required inputs are filled in accordance to the expected format. By default, JavaScript validation executes when the submission button of a form (i.e. executing the HTTP POST/GET method) is clicked. Additionally, there is an option to trigger validation as soon as the user proceeds to fill-in subsequent fields. This approach, also termed as Ajax (Asynchronous JavaScript and XML) is considered more efficient than the default approach as the user is notified immediately of an incorrect or missing input.

Overtime, it became clear that client-side validation on its own does not yield a sound security mechanism. Furthermore, since languages like JavaScript executes only at the browser level, users have full control on its execution as the validation can be turned off by disabling the JavaScript option in the browser. However, disabling JavaScript may cause some sites not to function properly as these sites mandate the use of JavaScript in order for their applications to function.

Server-side validation, on the other hand, is done at the application's web server, where completed forms must be sent over to the application's web server before input validation takes place. Unlike client-side validation, the user is able to submit an incomplete form without being prompted instantaneously. The data will only be analysed at the server end, and if there is an incorrect or missing input, the server returns the form with the highlighted errors. Server-side validation has a stronger ability to protect against input manipulation attacks disallowing bad data from being stored in the database. Server-side validation can be programmed using any of the web development/scripting languages like PHP, .Net, Java, Perl, and others.

Here is a sample web form containing client-side validation using *JavaScript*:

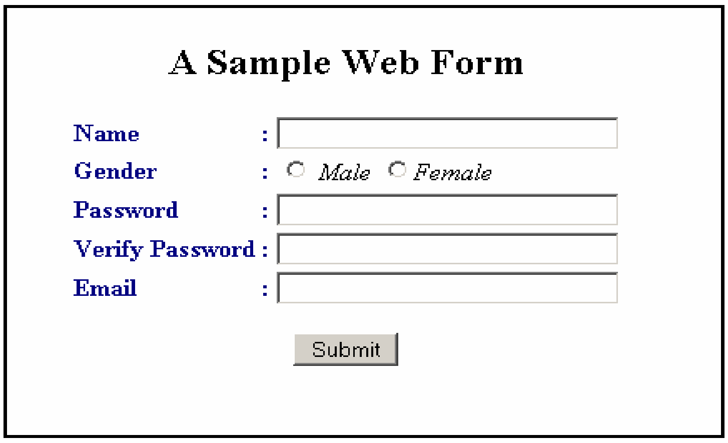


Figure 6: A Sample Web Form

Using the sample code in Figure 7, possible attack scenarios are as follows [3]:

1. ***Assuming that the* method *is set to* POST**
   (<form name="register" method=post action="test.html" onSubmit="return Validate(this)">)

   A. An attacker modifies the details of the source file in the following manner
      a. Save the source file (HTML embedded with JavaScript) on local
      b. Make the intended changes (e.g. change the value of the 'maxlength' variable

      ```
      Name : <input type=text name="name" size=30 maxlength=50>
      ```

      c. Modify the action attribute to contain the complete URL from which the form came from

      ```
      <form name="register" method=post action="www.website.com/test.html">
      ```

      d. Save the file on local
      e. Open the local file in the browser and submit the form

   B. Alternatively, since scripts are part of the DOM (Document Object Model), an attacker can for instance re-write or remove the *Validate(theForm)* function. In this way, the form can be submitted even with empty fields, bypassing the necessary validation.

      ```
      JavaScript:void(Validate(theForm)=new Function(alert(\'Validation by-
      passed!!\');document.forms[1]/submit();'))
      ```

      DOM is a model, more precisely, a platform and language-neutral interface, developed by the World Wide Web Consortium (W3C) to allow programs and scripts to dynamically access and update the content, structure and style of web pages. The DOM makes elements of a web page available as objects for scripting, particularly useful when there is a combination of HTML, style sheets and scripts required for interactive web pages [61].

```
<html>
  <title>Web Form</title>
    <head>
      <script language="JavaScript" >
        <!-- Begin
                function echeck(str) {
                /* Ensures the proper occurrence of the @ and . symbols in an email which
                should otherwise return an “Invalid E-mail” message” */
                }
                /* This part of the validation is for ensuring non-empty fields */
                /* Not all of the field validation are shown in this example */
                function Validate(theForm) {
                        if (theForm.name.value == "") {
                                alert("Please enter your Name");
                                theForm.name.focus();
                                return false;
                        }
                /* skipped validation for the Gender input */
                        // The following verifies the Password and Verify Password input
                        if (theForm.pswd.value != theForm.vpswd.value) {
                                alert("Password mismatch. Please verify the Password");
                                theForm.vpswd.value=""
                                return false;
                        }
                        if ((theForm.email.value==null)||(theForm.email.value=="")) {
                                alert("Please enter your Email")
                                theForm.email.focus()
                                return false
                        }
                        //The following validates the email address per the echeck function
                        if (echeck(theForm.email.value)==false){
                                theForm.email.value=""
                                theForm.email.focus()
                                return false
                        }
                        return true;
                }
           //  End -->
        </script>
    </head>
    <body OnLoad="reset ()">
        <p><center><b>A Sample Web Form</center>
          <br><form name="register" method=post action="test.html" onSubmit="return
Validate(this)">
             <center><table>
               <tr>
                  <td><b>Name</td>
                  <td><b>:</td>
                  <td><input type=text name="name" size=30 maxlength=20></td>
                  /* Not all of the HTML code are included*/
              <tr>
                  <td><b>Email</td>
                  <td><b>:</td>
                  <td><input type=text name=email size=30></td>
              </tr>
           </table>
        <p><input type="submit" name="submit" value="Submit"></center>
       </form>
    </body>
</html>
```

Figure 7: A Sample Web Form with JavaScript validation

C. An attacker may use a web proxy[1] to modify contents of pages traversing between the web browser (client) and the web server (server). A proxy server can be placed either in the user's local computer, or at specific points between the user's local computer and the destination servers. All data passing through the web proxy are mostly in unencrypted form. For this reason an insecurely configured web proxy can capture and record all data bypassing the proxy including unencrypted logins and passwords.

Some may argue that certain websites mandate the use of JavaScript in the browser in order to proceed with particular transactions. However, the above mentioned methods may be executed regardless of whether the JavaScript option is turned on or off.

2. ***Assuming that the* method *is set to* GET**
   (<form name="register" method=get action="test.html" onSubmit="return Validate(this)">)

   A. An attacker picks fields from the form, makes the desired modification and appends the resulting URL in the address bar of the browser.

When comparing server-side and client-side validation, several advantages can be seen if both the techniques are used concurrently. Using server-side validation solely for input validation is considered to be very inefficient since each completed form will need to be sent over to the web application's server for validation. On the other hand, client-side validation on its own is insufficient to protect against unvalidated input threats. Hence to avoid propagation delay and to reduce unnecessary traffic to the web server, it would be more effective if the first run of validation is handled by the client while both the advance and business logic validation is handled by the server. Business logic validation concerns the manner in which information is being handled between a database and a user interface. It prescribes how business objects like accounts, loans, and inventories interact with one another.

Nonetheless, the combination of server-side and client-side validation causes an increase in development time, and the duplication of logic might be considered to outweigh the advantages stated above [74].

The study of threats caused by unvalidated input proves its importance in implementation. A large number of attacks can be circumvented if developers pay close attention in complying with security standards and rules for input validation. As will be seen more clearly in subsequent chapters, all types of inputs in web applications must be consistently validated for type, length, format and range.

The following subsections describe the attacks made possible because of unvalidated input fields.

2.1.1 Buffer Overflows

A buffer is a temporary data storage location in memory. A buffer overflow occurs when data surpassing the boundaries of a defined buffer length is written into memory. If a program does not check the length of an input before entering memory space of either a stack or a heap, a buffer overflow could occur [19]. The overflow leads to overwritten stacks, variables, and/or memory addresses causing the application to crash, produce incorrect results or operate in malicious ways.

A typical example of a buffer overflow in web applications is when an attacker is able to manipulate an unvalidated input field, overwriting the execution stack[2] or stack pointers[3] of the web application, crashing the application web server, redirecting the program to execute malicious commands,

[1] A web proxy like a proxy server is an intermediate server residing between the client and the actual server and focuses exclusively on WWW traffic. It has the ability to alter the request/response coming from either the client or the server and in some instances service the client's request without contacting the specified server.
[2] An execution or call stack is a stack which stores information about the subroutines of an application.
[3] A stack pointer is a register which contains the last address of a stack of addresses.

changing program variables or revealing secure information. Conventionally, buffer overflow attacks occur when an attacker decides to modify an unvalidated URL by inserting a large string of characters like

http://www.bufferoverflow.com/main.cgi?username=xxxxxxxxxx........./

Inserting lengthy inputs can cause a web application or its backend database to malfunction. It is also possible to cause a denial of service attack against the web site, depending on the severity and nature of the flaw [14]. Therefore, without proper boundary checking, an attacker can attempt input that consists of executable code for the target system to run, along with a new return pointer for the stack [19].

The three common types of buffer overflow in web applications are stack based, heap based and off-by-one error based attacks. The most popular of them is the stack based buffer overflow which exploits the part of memory referred to as a *stack*. The stack is normally used to store inputs received from users. It is a data structure that works on the principle of Last-In-First-Out (LIFO); the last item put on the stack is the first item that is taken off [18]. Once user input has been received, the program than writes a return memory address to the stack to which the user's input is then placed above the memory address. When the stack is processed, the user's input gets sent to the specified return address, causing a buffer overflow when the user's input is either longer than the amount of space reserved or contains malicious content/command [14]. As a countermeasure, it is essential to reserve a sufficient amount of space for user input on the stack.

On the other hand, for an executable command to take effect, the attacker must be knowledgeable enough to specify a return address that points to the location of the malicious command. In order to know the exact memory range of the stack, the malicious command is often padded on both sides with NOP (no-operation) commands that basically does nothing but waste CPU clock cycles. However, if the address specified by the attacker falls within the padding range, the malicious command will be executed. Therefore, it is important to note that an executable command need not necessarily mean that the command will be executed.

The heap based overflow is similar to the stack based overflow except that the overflow occurs on the heap instead of the stack. Different from stacks, heaps are memory parts dynamically reserved by applications at run-time to store data. Heap based overflow attacks are considered to be rarer then the others with respect to the difficulty and complexity levels involved in causing overflows.

Lastly, the off-by-one error also known as array indexing errors is a growing concern within the buffer overflow category. This error occurs for instance when a user inputs an array with a different number of elements than is expected by the application and the bounds of the array are not checked correctly [26]. As an example, suppose a user provides an input *j*=5 for an array of size 5, and the code does not detect that the input value of 5 would yield an error. This error is known as the off-by-one error. The attempt to refer to an element outside the legal range of indices for that array is called the off-by-one error.

```
// creates an integer type array of size 5
int[] value = new int[5];

//user input is received and stored in a variable j (j =5)
for (int i = 0; i <=j; i++)
{
            System.out.println( value[i] );
}
```

From this study, it is better understood why, according to OWASP, it is not easy to discover and if discovered, difficult to exploit buffer overflow vulnerabilities in web applications. A buffer overflow attack is hard to carry out unless the attacker has access to the application's source code or memory dumps or has the ability to reverse engineer the application binaries [12, 25]. This perhaps is the reason why till date there has been no buffer overflow attacks in web applications reported by the Web Application Security Consortium [13].

2.1.2 Cross Site Scripting (XSS)

Cross Site Scripting (XSS) is a popularly growing method to seize private and confidential information from web application users. The term *cross site* refers to the manner in which the attack takes place, i.e. a user believes that it is corresponding with a trusted web server while in fact it is simultaneously providing information to a fraudulent server. [21] highlights that XSS is found to be the most common type of injection flaw, with seven out of 10 web applications vulnerable to this type of attack. Also, it indicates that more than 70% of all web forms are said to be XSS vulnerable.

In simplicity, XSS allows an attacker to gain elevated access to inject malicious scripts into the input fields or URL parameters that have not been validated and consequently affecting users that visit the unsuspicious web pages/links. Successful XSS attacks may cause disclosure of user's session cookie or files compromising confidential information, installation of Trojan horse programs, diversion of user information to unauthorised or unauthenticated devices, modification of trusted web page contents to display false information and other related risks [27, 30].

An XSS attack is easily enabled because web servers do not have the ability to distinguish between data supplied by the user and data supplied by the source of the form before being filled by the user. Most dynamic web applications normally request information from users and using the received information attempts to display the correct content to the user. In some cases, however, the output is improperly generated or certain malicious scripts is executed because the accepted input was not properly validated or encoded, causing an XSS attack.

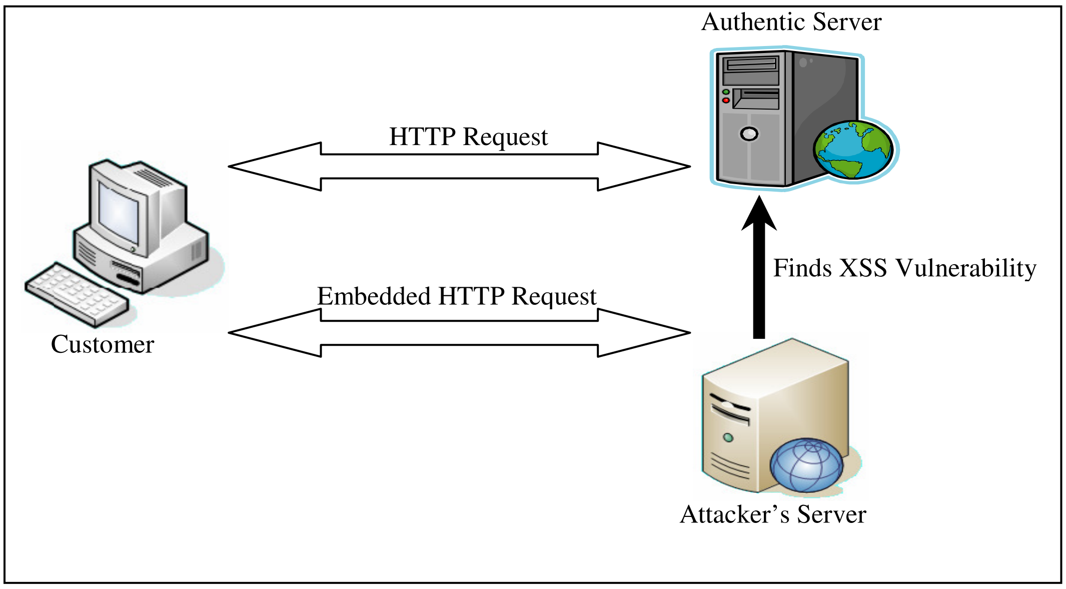


Figure 8: Cross Site Scripting

Currently, there exist mainly three ways in which an attacker uses XSS to deceive both novice and naive users;

i. Stored
ii. Reflected
iii. DOM-Based[4]

[4] Document Object Model (DOM) : is a platform and language independent standard object model for representing HTML or XML and related formats. (Taken from Wikipedia)

i. ***Stored***

User inputs (malicious or otherwise) received from unvalidated input fields are often permanently stored in a database or file system residing on a web server. The stored information are later retrieved and displayed just like in forums, message boards, blogs and the like. Threats arise because the collected data is not properly validated and are being displayed without appropriate HTML character encoding.

This form of XSS is regarded to be the most powerful of its kind.

Example

An attacker posts a message (HTML formatted) on a message board containing a link to a seemingly harmless site, which subtly encodes a malicious script that attacks users who clicks on the link. Such an attack can cause adverse impact on a large number of users especially if the link is infected with either a cross site scripting virus, Trojan horse program or several other malicious/executable commands leading to damages like Distributed Denial of Service (DDOS), spam and dissemination of browser exploits [20].

For example, an attacker can hide a malicious script in an anchor as shown below

```
<A HREF="http://testXSS.com/malScript.html?script=<SCRIPT SRC=
'http://testXSS.com/js/malScript.js'></SCRIPT>">Most Watched Movies </A>
```

This link would send a user to *http://testXSS.com/malScript.html* which will then execute the *malScript.js* script while being loaded.

ii. ***Reflected***

Inputs or requests received are immediately parsed by the web server in order to generate subsequent web pages. Hence, if the inputs supplied are not properly validated the resulting web page is generated without appropriate HTML encoding allowing client-side scripts to be executed when the page loads. Although normally users have full control over what is being inserted in the input fields, an attacker applying some social engineering skills could embed a malicious script into the CGI parameter of a legitimate website. The tampered URL is sent to potential victims via email, chat rooms, search engines, bulletin boards and others just to name a few.

Even though this form of XSS is regarded to be the most common and well-known type but since it requires some social engineering abilities, this form of XSS is not regarded to be as harmful as the *Stored* version.

Example

An attacker creates a mirrored design of a trusted web site and fixes the values of certain input fields in advance or includes malicious scripts or commands with the intention of either gaining unauthorised information by requesting security credentials or corrupting the application or just damaging the user's system. Upon accessing the supposedly trusted web site (e.g. via spoofed emails), the malicious script/command rooted on the web site executes. The script for instance is used to steal sensitive information to be stored on the attacker's computer without the user's knowledge.

iii. ***DOM-based***

DOM-based XSS vulnerabilities exist within a page's client-side scripts itself. This form of XSS is similar to the Reflected method except for the fact that it has additional effect of remote execution vulnerabilities [18, 25].

DOM-based XSS is mainly caused by HTML pages that use JavaScript to execute commands like *document.location* or *document.URL* or *document.referrer* in an insecure manner [31]. Generally, DOM (Document Object Model) in HTML is a standard set of objects used to access and manipulate HTML documents. All HTML elements which make up the contents of a web page can be accessed, created, modified and deleted using the DOM objects. An example of a DOM object is the document object, specifically applicable for XSS. The document object represents an entire HTML document and can be used to access all elements in a page from a script embedded in the HTML document. This document object contains several pre-defined page properties (location, URL, referrer), each having its own functionality but at the same time having some flaws pertinent to XSS attacks.

JavaScript, the most common and browser compatible client-side scripting language, is well known for its use of objects which are represented as DOM in web browsers. Each object in JavaScript can have various properties, methods and event handlers.

Example
Consider the following DOM-based XSS analogy [31]. Supposing the web address is http://www.vulnerable.site/welcome.html and the HTML code, embedded with JavaScript is as follows:

```
<html>
<title>Welcome!</title>
<script>
Var pos=document.URL.indexOF("name=")+5;
document.write(document.URL.substring(pos,document.URL.length));
</script>
<br>
Welcome to our system
.
.
.
</html>
```

A user named Joe, who accesses this web page, will have the address read as
http://www.vulnerable.site/welcome.html?name=Joe

However, if the link above is inserted with the following JavaScript,
http://www.vulnerable.site/welcom.html?name=<script>alert(document.cookie)</script>

an XSS breach has then occurred. How? The victim's browser starts parsing this HTML into DOM. When the parser arrives to the JavaScript code, it executes it and modifies the raw HTML of the page. Similar attacks may be executed remotely. For instance, a central image, controlled by a timing function will cause an image to change after a certain point and could potentially lead to XSS worms' exploitation.

```
<script>
image_1 = new Image();
image_1.src= "http://ha.ckers.org/images/kcpimp.jpg";
if (image_1.width > 1 ) {
// Execute malware
}
</script>
```

If the image is over a certain height or width or the combination is just right, the JavaScript will execute malware. Malware, depending on the application, may be something as simple as stealing a user cookie to a complex action of loading up an executable virus.

The most common web components that fall victim to XSS vulnerabilities include CGI scripts, search engines, interactive bulletin boards, and custom error pages with poorly written input validation routines [27].

It now becomes even clearer on how important it is to validate inputs received and encode outputs sent over to client browsers. Cenzic Inc., in its 2007 Quarter 1 Application Security Trends Report, highlights the Top 10 vulnerabilities in commercial and open source web applications. Of the reported vulnerabilities, file inclusion, SQL injection, cross site scripting and directory traversal were the most prevalent, totaling 63% of all web application attacks. The majority of vulnerabilities affected web servers, web applications and web browsers [21]. This further prompts for a thorough input validation techniques including appropriate filtering and encoding mechanisms.

2.1.3 Injection Flaws

Injection flaws allow attackers to relay malicious user supplied data through a web application to another system [25, 28] with the intention of obtaining unauthorized access to protected data or executing malicious commands to subvert the application. As will be seen, SQL injection is considered to be a subset of injection flaws.

Injection happens when unvalidated input is being passed through HTTP requests. An attacker can exploit unvalidated input fields by injecting special (meta) characters, malicious commands, or command modifiers to which the web application will blindly execute [25]. A successful injection attack leaves a damaging effect of accessing, corrupting and destroying confidential contents or destroying the entire application.

One example of an injection flaw is when an attacker modifies an input parameter that would modify the actions taken by the application. For instance, instead of retrieving the current user's file, the modified input makes a request to access another user's file (e.g., by including path traversal "../" characters as part of a filename request). This type of injection is also termed as parameter manipulation or parameter tampering.

When a web application does not properly sanitize input before using it, it may even be possible to trick the application into executing operating system or shell commands [35]. This can be done, for example using the pipe symbol as shown below.

Assuming that the original URL of a web application is

```
http://example/cgi-bin/showInfo.pl?name=John&template=tmp1.txt
```

An attacker can trick the web application into executing the command /bin/ls:

```
http://example /cgi-bin/showInfo.pl?name=John&template=/bin/ls|
```

In most cases, parameter manipulation is as simple as changing a variable in a form and having unauthorised data returned. It allows a malicious user to alter the data sent between the browser and the web application server [34]. In a poorly designed web application, malicious users can modify parameters like prices in web carts, session tokens or values stored in cookies, hidden fields and even HTTP headers. Therefore, no data sent or received from the client can be relied upon to stay the same unless cryptographically protected at the application layer [25]. Cryptographic protection using the SSL/TLS technology only protects data during transmission and in no way protects against parameter manipulation before being transmitted.

Injection flaws can be executed in:

- Cookies
  Cookies store information related to user preferences and session maintenance. An attacker can easily manipulate data residing within a cookie by modifying the cookie at the client's end or while the cookie is being sent to the server.

- Form Fields
  The example for this is as elaborated in Chapter 1 which shows how HTML source code embedded with JavaScript client-side validation can be saved, modified and sent to the application server, bypassing validation.

- URL/Query Strings
  Whenever completed HTML forms are sent over to the web server, occasionally, the URL of the webpage will contain all of the form field names and their respective values, which can be easily manipulated. This is especially true when the GET method is used.

- HTTP Headers
  HTTP headers consist of control information namely HTTP requests (from client to server) and HTTP responses (from server to client). Since the HTTP request headers originate from the client, it is susceptible to modification.

It is of good practice to never trust any input that originates or returned from the client.

2.1.3.1 SQL Injection

Databases in web applications are used for a variety of purpose; generally to store and retrieve information dynamically. For this reason, it is crucial that at all instances, a web application is able to provide accurate results based on queries made by users. It is more important, especially in aspects of trust, that the information (stored/retrieved) from a database maintains a high level of user confidentiality and integrity. This is essential in applications relating to defense, finance, medical and any other private data.

With the increase transition towards dynamic web applications, another threat, due to unvalidated input field arose: SQL Injection. SQL injections transpire due to intermixing of user supplied data with dynamic database queries or poorly constructed stored procedures. A stored procedure is a set of SQL statements stored in the database server. Different from a database query, when using stored procedures, applications need not make individual database accesses for retrieving or storing data but can reuse the statements in the stored procedures instead. The MySQL Developers guide [80] outlines the reasons why stored procedures are found to be more beneficial than conventional database accesses.

- Useful when multiple client applications, written in different languages or work in different platforms are to be integrated.
- Stored procedures are best used for all common operations. This provides a consistent and secure environment, and routines can ensure that each operation is properly logged. In such a setup, applications and users would have no access to the database tables directly, but can only execute specific stored routines.

Nonetheless, there should be a strict standard or rules governing the way in which stored procedure routines are written in order to prevent probable SQL injection attacks. This is further explained in both Chapter 3 and Chapter 4 of this thesis.

Contrastingly, conventional database queries in web applications allows users to directly access contents of a database, giving way to unethical users to manipulate the SQL queries, opening avenues for SQL injection vulnerabilities.

SQL injections enable attackers to create, read, update, or delete any arbitrary data sometimes completely compromising the application's database system as well as all other systems integrated with it. In its 2007 first quarterly report, Cenzic Inc highlights that roughly two out of 10 web applications were found to be vulnerable to SQL injection attacks. This attack can be carried out without requiring prior knowledge of neither the application nor the source code. Moreover, the task is easier since most web applications use the Structured Query Language (SQL) to interact with databases. Therefore, attackers can take advantage of common developer flaws to gain some insights before completely damaging the integrity of an application's database.

By using the input fields of a web form, an attacker inserts malicious SQL code to gain access to resources, modify existing data, sometimes even download an entire database or interact with it in illicit ways [32]. To begin an attack, an attacker enters a single quote into an input field (e.g. email address) with the intention to check if the input is being validated. Suppose upon submission, the session aborts with a syntax error, i.e. '500: Internal Server Error'. This will indicate that the SQL parser found the quote. In other words, the inputs are being parsed literally. The error message is a dead giveaway that user input is not sanitized properly and the application can be exploited [22]. Caleb Sima, who is the CTO of SPI Dynamics, Inc., an organization dedicated towards providing security solutions for web applications, commented that potential automation of SQL injection attacks gives rise to the possibility of a SQL injection worm.

Below illustrates two examples of an SQL injection attack.

Example 1

a. Suppose an attacker sees a URL like [33]

```
http://www.mydomain.com/products/products.asp?productid=123
```

Then, the attacker would be able to guess the likely SQL query

```
SELECT ProductName, ProductDescription FROM Products WHERE
ProductID = 123
```

Clearly, this example shows that the ProductID parameter is passed to the database without sufficient protection allowing an attacker to exploit the vulnerability.

b. The attacker can manipulate the parameter's value (i.e. modifying the ProductID value to read "123 OR 1=1") by making the following changes on the URL

```
http://www.mydomain.com/products/products.asp?productid=123
or 1=1
```

This will return all pairs of ProductName and ProductDescription associated with the stated ProductID.

c. An attacker may also retrieve data from other tables within a database by using the SQL UNION SELECT statement. The UNION SELECT statement allows the chaining of two separate SQL SELECT queries that have nothing in common.

```
http://www.mydomain.com/products/products.asp?productid=123
UNION SELECT username, password FROM USERS
```

The execution of the URL displays a table with two columns, containing the results of the first and second queries, respectively

```
SELECT ProductName, ProductDescription FROM Products WHERE
ProductID = '123' UNION SELECT Username, Password FROM Users;
```

Example 2 [62]

a. Figure 9 is an example of a typical login site where a user is required to enter his username and password in order to log in

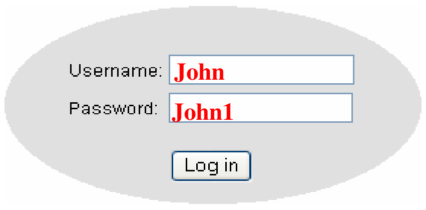


Figure 9: SQL Injection example (a)

The SQL query that follows from here is usually of the following form

```
$result = mysql_query(
          "SELECT * FROM Accounts " .
          "WHERE Username = 'John' " .
          "AND Password = 'John1';");
if (mysql_num_rows($result) > 0)
          $login = true;
```

b. Suppose an attacker inserts the following

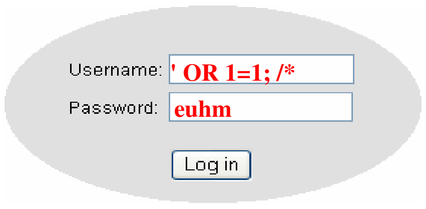


Figure 10: SQL Injection example (b)

The SQL query executed would be

```
$result = mysql_query(
          "SELECT * FROM Accounts " .
          "WHERE Username = '' OR 1=1; /*' " .
          "AND Password = 'euhm';");

if (mysql_num_rows($result) > 0)
          $login = true;
```

Since the statement "**1=1**" is always true (or in fact any mathematical operation that is a tautology), the query would return all usernames existing in the database. The **/* (sometimes also -- or #)** comments out the portion "AND Password = '**euhm**';". Hence, any data appearing in the query after the comment will be ignored.

c. In the following example, the attacker attempts to erase the entire table by executing the command in Figure 11. The semicolon (;) in the example is used to pass multiple statements to the database server in a single execution. Hence, the subsequent statement "DROP TABLE FinData" causes the SQL Server to delete the entire FinData table.

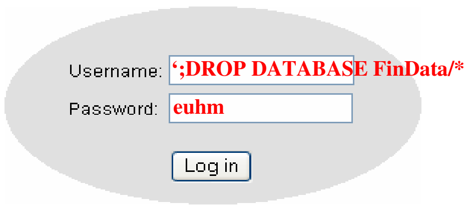

Username: ';DROP DATABASE FinData/*
Password: euhm

Figure 11: SQL Injection example (c)

As was seen, successful SQL injection attacks requires the understanding of the underlying database. An attacker usually prevails through trial and error, especially if the application ignorantly reports any database errors to a patient attacker.

2.2 Broken Access Control

Access control commonly known as authorization concerns the access to contents and/or functions entitled to users of an application. Different from authorization is the concept of authentication which refers to the process of validating a user based on who they say they are. Naturally, an authentication process is followed by an authorization process.

Applications requiring access control usually groups its users according to roles which distinguish privileges according to their various responsibilities. There are many different types of access control models available to aid developers in implementing access control functionality in applications. From an implementation standpoint, access control is the part of code that restricts access to certain functions of an application by requiring users to first be authenticated for access. Popular access control models are Mandatory Access Control (MAC), Discretionary Access Control (DAC), and Role-Based Access Control (RBAC). There are plentiful resources on the Internet that describes these models in detail. RBAC has become the predominant model for advanced access control because it reduces the complexity and cost of security administration in large networked applications. However, depending on the application, developers have the choice of selecting the most appropriate model. It is the responsibility of application developers to ensure that proper authorization is performed by the program before granting access to sensitive information.

A weak or broken access control mechanism may lead to undesired/damaging situations where protected contents are compromised (i.e. viewed, changed or deleted by unauthorised parties) or enables attackers to carry out unauthorised functions further impeding the security of the application.

A simple example: Let there be a web application which has an additional message sending functionality. Using this functionality, users can read e-mail correspondences with regards to services requested or provided by the application or organization hosting the application. It is imperative that users should only be able to read their own correspondence.

Given a URL like:
https://vulnerable.com/messaging/message-viewthread.jsp?selectedListItemId=6100&listname=messageThread&listHandlerRegistryKey=messaging/message-viewlist.jsp_messageThread

a user might gain unauthorized access to email contents by, for instance changing the number (6100) stated in this URL. By exploiting this vulnerability, users can gain unauthorized access to some or all messages that are sent/exchanged within the application which may include messages informing users about their passwords etc.

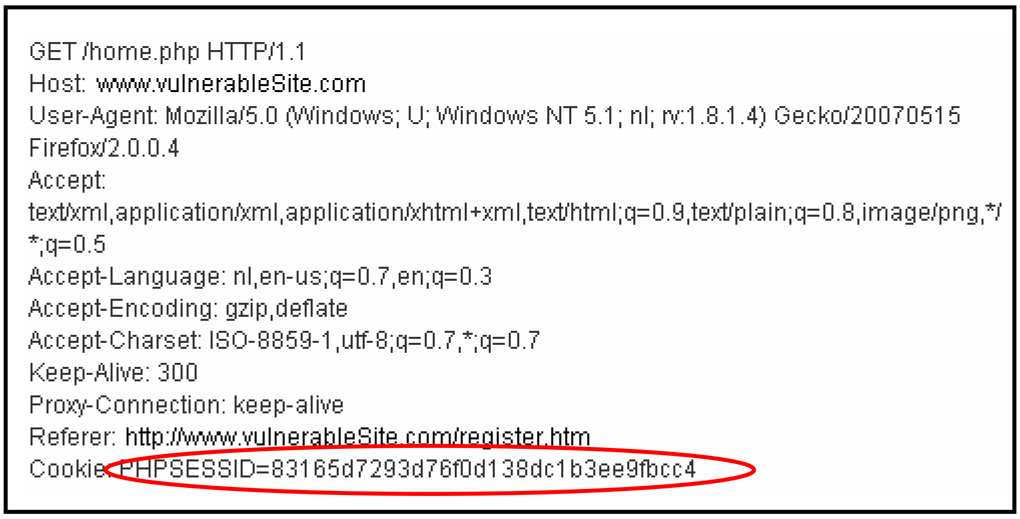
In Chapter 3, a more comprehensive example, detailing how access control mechanism can be implemented in Java and .NET is discussed and elaborated.

2.3 Broken Authentication and Session Management

Authentication in web applications commonly involves the use of credentials like usernames and passwords. It is used to verify the identity of a user or if the user is who or what it claims to be. Following a successful authentication and authorization process, is the initialization of sessions. Sessions are used to keep track of the stream of requests received from each user [25]. Together, authentication and session management includes all aspects of handling user authentication and managing active user sessions principally ensuring that user's credentials (stored or in transmission) are protected at all times. Alongside, came the credential management functions which include functions like changing passwords, retrieving forgotten passwords, remembering passwords, and other related functions.

With reference to implementing a strong authentication and session management scheme, it is essential to outline a policy document detailing the expectation of the application. For instance, exerting complex password rules, limiting failed login attempts, not storing/retrieving passwords in clear-text form, performing additional authentication via HTTPS/SSL, transmitting session IDs via HTTPS/SSL, avoiding the use of session IDs in the URL, encrypting sensitive information stored in sessions, and others to name a few. Consistent adherence to such policy is believed to lead to a secure and robust authentication and session management mechanism.

Weak authentication and session management is subject to various attacks including password guessing, sharing, cracking and sniffing. Consequently, a compromised password opens a whole new avenue of application insecurity thus potentially losing user's trust in an application.



```
GET /home.php HTTP/1.1
Host: www.vulnerableSite.com
User-Agent: Mozilla/5.0 (Windows; U; Windows NT 5.1; nl; rv:1.8.1.4) Gecko/20070515
Firefox/2.0.0.4
Accept:
text/xml,application/xml,application/xhtml+xml,text/html;q=0.9,text/plain;q=0.8,image/png,*/
*;q=0.5
Accept-Language: nl,en-us;q=0.7,en;q=0.3
Accept-Encoding: gzip,deflate
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7
Keep-Alive: 300
Proxy-Connection: keep-alive
Referer: http://www.vulnerableSite.com/register.htm
Cookie: PHPSESSID=83165d7293d76f0d138dc1b3ee9fbcc4
```

Figure 12: Example of an insecure Session ID

2.4 Improper Error Handling and Logging

This category of vulnerability requires the understanding of three different concepts; Error handling, Exception handling and Logging that must be incorporated in any application programming. An error is an irrecoverable condition that occurs during runtime. Examples of errors in a program are OutOfMemory, StackOverflow, Server errors, etc. Error handling anticipates unexpected errors and recovers without the application terminating or crashing and if it's perceived to be severe, the application is able to gracefully terminate.

Different from errors are exceptions. Exceptions are events that occur during the execution of a program that disrupts the normal flow of the program's execution. Running out of disk space on a file or database server which results in an update failure is one example of an exception. In most cases, exceptions can be thrown (predicted), caught (detected) and repaired (recovered). It provides the application with recovery functions when the main logic of a problem is challenged.

Together, error handling and exception management refer to the ability of an application to accept occasional failures while at the same time having the capability to deal with the failure in the most secure manner possible, but still maintaining core functionality of the application [36].

Whether during the development, testing or maintenance stage of an application, error/exception handling helps developers by providing sufficient amount of information about a certain unexpected event that took place within the application. Hence, it is very important that, prior to the development stage, application designers and developers take precautions in designing and implementing intelligent error messages caused by possible failure conditions at each process, module or data store without revealing the application, database and/or server structure. The error message must be meaningful enough to the user and adequate enough for site maintainers to make the necessary changes. Another approach is to hierarchically walk through the functionality of the application in the same way a user might progress through multiple areas of the system [36]. Additionally, all error and exceptions should be logged so that any hacking attempts can be traced and analysed.

Every piece of information an attacker receives about a targeted system or application becomes a valuable means to launch an attack. Furthermore, the manner in which web applications normally handle errors or exceptions makes it easily susceptible to all kinds of threats and attacks. At times, these error messages may contain important debugging or implementation details such as stack traces, database dumps, out of memory, null pointer exceptions, system call failure, database unavailable, network timeout and error code which otherwise should remain restricted. This leaked information may provide attackers with a strong base in planning their attack.

Improper error/exception handling leads to implications like system crash, system consuming significant resources, or effectively denying or reducing service to legitimate users [25]. Some examples, extracted from [23, 25] are as follows:

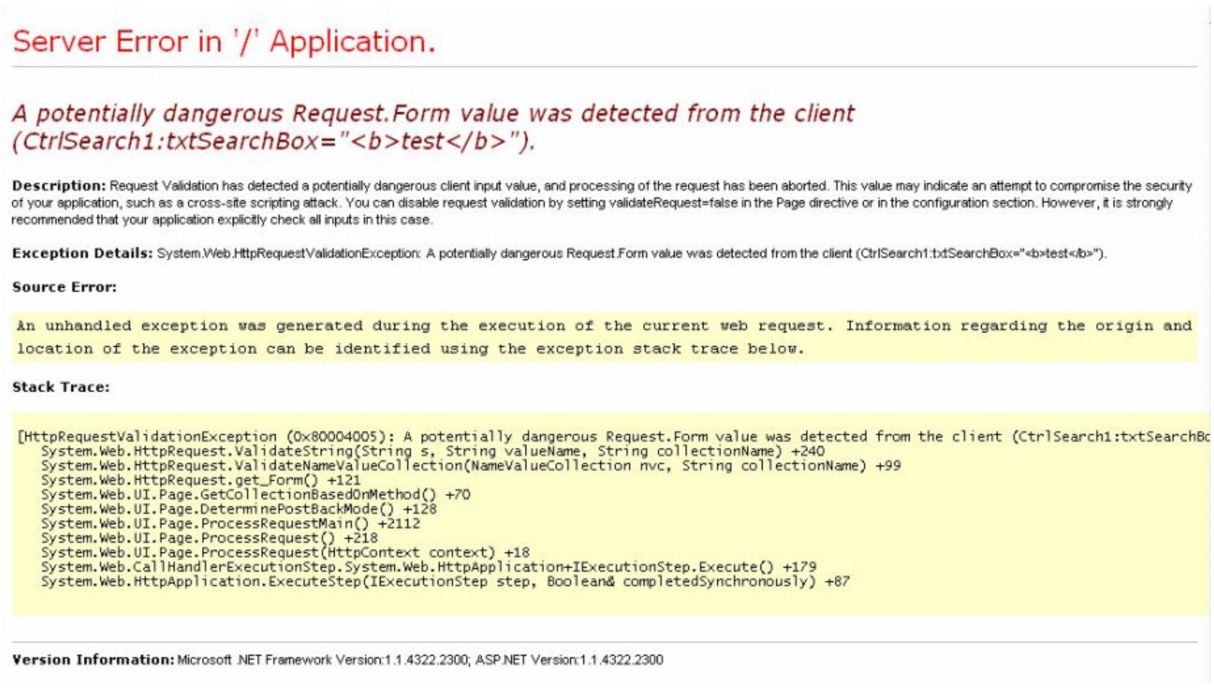
- Error messages like 'invalid username', 'invalid password', 'directory or file not found', 'permission to view this directory or file is restricted', gives away valuable hints about the system's internal components including information about the database, table and field names. A real life example of such errors is the Bugzilla error page which unknowingly displayed the database password in an error message when the SQL server was not running.

- Errors that supply the full path name to executables, data files and other system assets help the attacker understand how things are laid out behind the firewall, sometimes revealing the component of interest.

- Attackers can make use of error conditions that consume system resources such as CPU and disk to create, for instance a denial of service (DoS) attack.

From the description of this threat, it is important to learn not to provide details when formulating error handling procedures for users. Even a small amount of data leakage occurring via error handling can be extremely dangerous [23]. In order to thwart attempts of attackers who try to gain knowledge about an application through its error handling messages, it is imperative to append a detailed logging and auditing mechanism which would help ensure accountability. Furthermore, the awareness of a possible logging mechanism acts as a deterrent for attackers to start an attack.

Notifications are also necessary to help administrators to be aware of undesired events occurring in an application. Notification rules can vary based on the criticality of the failure. For instance, authentication failure due to a denial of service attack requires more attention than a specific user whose account is being brute-forced [36]. Logging, notification, and periodic auditing are still insufficient for a complete error handling, exception management and logging scheme. There is the notion of performing cleanup. Cleanup involves reclaiming of resources, rolling back of transactions or some combination of the two [36].


Server Error in '/' Application.

A potentially dangerous Request.Form value was detected from the client (CtrlSearch1:txtSearchBox="<b>test</b>").

Description: Request Validation has detected a potentially dangerous client input value, and processing of the request has been aborted. This value may indicate an attempt to compromise the security of your application, such as a cross-site scripting attack. You can disable request validation by setting validateRequest=false in the Page directive or in the configuration section. However, it is strongly recommended that your application explicitly check all inputs in this case.

Exception Details: System.Web.HttpRequestValidationException: A potentially dangerous Request.Form value was detected from the client (CtrlSearch1:txtSearchBox="<b>test</b>").

Source Error:

```
An unhandled exception was generated during the execution of the current web request. Information regarding the origin and
location of the exception can be identified using the exception stack trace below.
```

Stack Trace:

```
[HttpRequestValidationException (0x80004005): A potentially dangerous Request.Form value was detected from the client (CtrlSearch1:txtSearchBc
   System.Web.HttpRequest.ValidateString(String s, String valueName, String collectionName) +240
   System.Web.HttpRequest.ValidateNameValueCollection(NameValueCollection nvc, String collectionName) +99
   System.Web.HttpRequest.get_Form() +121
   System.Web.UI.Page.GetCollectionBasedOnMethod() +70
   System.Web.UI.Page.DeterminePostBackMode() +128
   System.Web.UI.Page.ProcessRequestMain() +2112
   System.Web.UI.Page.ProcessRequest() +218
   System.Web.UI.Page.ProcessRequest(HttpContext context) +18
   System.Web.CallHandlerExecutionStep.System.Web.HttpApplication+IExecutionStep.Execute() +179
   System.Web.HttpApplication.ExecuteStep(IExecutionStep step, Boolean& completedSynchronously) +87
```

Version Information: Microsoft .NET Framework Version:1.1.4322.2300; ASP.NET Version:1.1.4322.2300

Figure 13: Example of an error message that leaks information about a web application

2.5 *Insecure storage*

In any application, it is of paramount importance to have a secure storage system, whether on disk or in memory, especially where confidential (private) information are at stake. Most contemporary web applications collect and store information such as usernames, passwords, social security, account statements, medical history and various other proprietary information. The collected information must be kept in a highly secured storage area.

Encryption is a popular technique used to protect sensitive information stored either in a session/cookie, database or on a file system. Cryptographic means are used to strengthen the storage system of an application. While encryption via cryptography has become relatively easy to implement (with the availability of many libraries/algorithms and protocols) and use, developers still frequently make mistakes when integrating it into a web application. Moreover, ways to protect and handle the keys being used must be well thought of.

Like in all other threats, a break into an information storage area could compromise the confidentiality and integrity of information leading to insecure passwords, files, caches and configurations. Attacks can take place via an authentic account on an application, or by taking advantage of an operating system's vulnerability to gain access, or access to a process's memory via a core dump or a debugger, or by extracting sensitive information insecurely stored in a cookie or URL. Encryption via cryptography only makes it difficult (not impossible) and time consuming for intrusion attempts. According to [30], all the traditional cryptanalysis approaches can be used to attempt to uncover how a web site is using cryptographic functions.

```
geg[voornaam]□Johann□www.vulnerableSite.nl/□1024□2371656320□29885080□2136737680□29879045
□*□geg[achternaam]□Strauss□www.vulnerableSite.nl//□1024□2371656320□29885080□2136737680□29
879045□*□geg[inlognaam]□Hilversummer□www.vulnerableSite.nl/□1024□2371656320□29885080□213
6887680□29879045□*□geg[userid]□24□www.vulnerableSite.nl/□1024□2371656320□29885080□213688
7680□29879045□*□geg[check]□892b9eaf8b7f8e6470ec7829b04e3d0a□www.vulnerableSite.nl/□1024□23
71656320□29885080□2136887680□29879045□*□
```

Figure 14: An insecure storage of user information where firstname, username and userid is stored in plaintext

2.6 *Application Denial of Service*

A denial of service (DoS) in web applications is distinct from the popular DoS in networks. It is intended to prevent web sites from serving its users. Since there is no reliable way to tell where a HTTP request is coming from, it is very difficult to filter out malicious traffic [25]. Bots and scripts can be used to load an application with supposedly legitimate HTTP requests at a rate which consumes all available resources (such as bandwidth, database connections, disk storage, CPU, memory, threads, or application specific resources) on the web server. When any of these critical resources reach full utilization, the web site becomes inaccessible, denying service to its users.

A single attacker can generate excessive traffic on its own to swamp an application and when this happens, all authorized users start having difficulty in using the system. Besides, an attacker may also lock out a legitimate user by sending a series of invalid credentials until the system locks out the account. These attacks instantly prevent all other users from using the application.

Application DoS can be the result of vulnerabilities like SQL Injection, Injection Flaws, Access Control, Buffer Overflows and Error Handling and Logging. For instance, an attacker can use SQL Injection techniques to modify a database system by deleting/modifying data or dropping tables. An attacker can also send specially crafted requests that will crash the web server process. In both circumstances, the application will cease to provide proper services or become inaccessible to its users.

The following example, taken from the Web Application Security Consortium, describes a classic application denial of service attack.

A medical web site generates medical history reports for its patients. For each report requested, the web site queries the database to fetch all records matching a single social security number, for instance. Because the database holds an abundant sum of records, the user will need to wait for approximately three minutes, as the database searches for matching records, before their medical history report is returned. During the three minutes, the database server's CPU reaches 60% utilization.

An attacker attempting an application DoS attack will send 10 simultaneous requests to generate a medical history report. These requests will most likely put the web site under a DoS condition as the database server's CPU will reach 100% utilization. At this point the system will most likely be inaccessible to its regular users.

Thorough analysis of the application and its functionality can enable more sophisticated application DoS attacks which often are due to one of the existing vulnerabilities in web applications.

2.7 *Insecure Configuration Management*

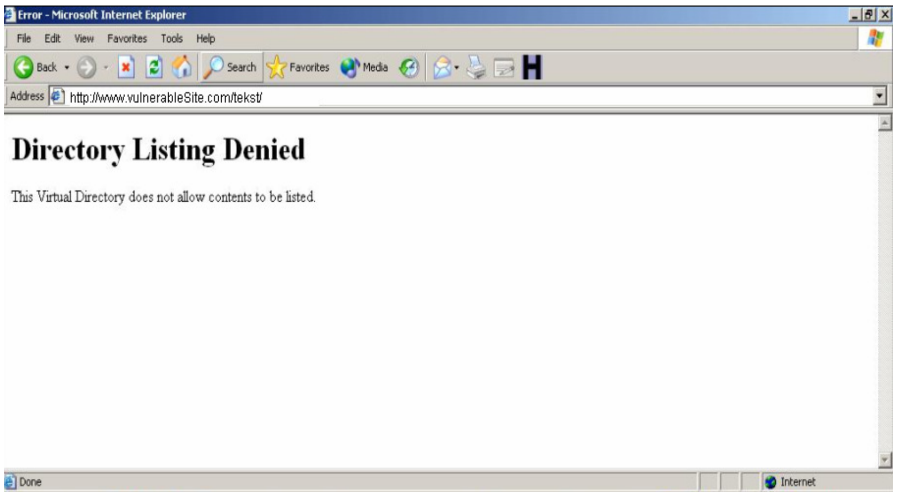
Industry analysts, especially Gartner and Forrester Research defines configuration management as the combination of various management disciplines to maintain a stable and desired state of a system. The principal goal of configuration management is to ensure all changes to the environment, in this context, the web application, are properly documented and tracked. Software Configuration Management (SCM) is the discipline whose objective is to identify the configuration of software at discrete points in time and to systematically control changes to the configuration for the purpose of maintaining software integrity, traceability, and accountability throughout the software life cycle [70].

In any web applications, the web servers, application servers and database servers are responsible for serving content, invoking applications that generate content, providing data storage, directory services, mail, messaging and various other services [25]. Therefore, a good configuration and control framework encompassing comprehensive web content change management and product versioning configuration scheme enables web applications to be organized in a logically structured manner.

As part of the effort, it is imperative to ensure that architects, developers and testers pay careful attention to configuration management issues during the design and development of an application rather than leave it to administrators at deployment [21]. Failure to manage can lead to a variety of security problems such as

- Flaws and mis-configurations permitting directory listing which eventually leads to directory traversal attacks
- Unnecessary default, backup, or sample files being easily accessible
- Improper file and directory permissions
- Unnecessary services enabled, including content management and remote administration

To ensure integrity and consistency, the change management and product versioning configuration scheme requires applications to store the history of all changes made to the application [27]. Furthermore, organizations have a variety of legal and regulatory compliances that ultimately result in requirements other than just tracking every change to critical data assets, but also associated requirements such as who made the changes to the data, what time a change was made, what changes were made, why the change made, etc. Hence, in the long run, a well-defined, secure and practiced configuration management process achieves optimization of IT assets, enables application changes to take place smoothly, resolves any potential problems in a more systematic way, and most importantly through secure configuration policies and practices, enhances the security of the application.


Error - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites Media
Address http://www.vulnerableSite.com/tekst/
Done
Internet

Directory Listing Denied

This Virtual Directory does not allow contents to be listed.

Figure 15: A secure configuration management will deny contents from being listed to regular users

Summary

The result of this chapter is a comprehensive analysis of the most reported vulnerabilities existing in today's web applications. The vulnerabilities discussed include input validation errors such as buffer overflows, cross site scripting, injection flaws, and SQL injection; broken access control, broken authentication and session management, improper error handling and logging, insecure storage, application denial of service and insecure configuration management.

The several attacking strategies show that the vulnerabilities are mostly due to implementation flaws. Some flaws involve more than one category of vulnerability. For instance, an unvalidated input can cause an injection flaw that leads to a buffer overflow that eventually crashes the application server causing an application DoS attack. Broken access control and insecure information storage leading to the compromise of confidential information is another example resulting from more than one vulnerability. With this in mind, the mentioned vulnerabilities should be given equal attention in the implementation stage of any web application.

When developing web applications, there are many details that must be considered. Attention must not be focused solely on the functionality of the application but also on protecting information that is captured, distributed and stored within the application. This shows that programmers must realise how their code plays a major role in enforcing security. Every line/function/class of code must be written carefully considering all possible threats surrounding the application. Secure coding plays a significant part in the development of secure web applications.

Having studied the various types of threats and vulnerabilities, we will now take a look at the available libraries, classes, frameworks and related components of the two widely used web development languages, i.e. Java and ASP.NET.

3 Available Prevention Mechanisms

There is a large number of programming languages available for today's web programming needs. The choice of a language is dependent on the goal and functionality of the application. Some of the criterions in choosing the most suitable programming language are [51]

- The ability to deal with a variety of protocols, formats and programming tasks;
- performance;
- security;
- platform independence;
- protection of intellectual property; and
- the ability to integrate with other web tools and languages

Logically, it is impossible for a language to fulfil all of the listed criteria in equality. For example, integrating the application with various vendor applications may raise questions about the protocols, performance and security of the application. Hence, some criterions usually take precedence over the other.

The three most popular web programming languages in use today are namely Java, .NET(ASP), and PHP. With the growing hype and emphasis of security in web applications, these languages are undergoing continuous improvement. This chapter aims to study and list the existing libraries, classes, frameworks and related components of the two widely used web development languages, i.e. Java and ASP.NET, focusing on their individual ability to withstand attacks prescribed in Chapter 2.

An inexperienced programmer, new to either Java or ASP.NET technologies for web development is advised to read Appendix B (Java) or Appendix C (ASP.NET) before proceeding.

3.1 Java

Besides its cross-platform nature, the Java language is designed to offer secure Internet communications through the use of its numerous built-in libraries. Some examples include the Java Secure Socket Extension (JSSE), Java Advanced Intelligent Network (JAIN), Network Security Services for Java (JSS) and others.

In the same way, from an application viewpoint, Java is said to have a number of built-in security features to support the distribution of applications across the Internet. The following subsections discuss the various libraries, classes, frameworks and other components in Java that render secure web applications.

Each subsection ends with a ***Discussion and Conclusion*** box where we discuss the strength and weaknesses of the available mechanisms. This will hopefully result in more value of the information given in practice.

3.1.1 Input Validation

Deriving from Chapter 2, the criterions for input validation can be summarised to

i. Ensure that all user inputs or data received are captured in its right type, length, format and range
ii. Prevent users from entering incorrect or unacceptable values

The following lists some of the prevention mechanisms in Java to achieve the aforementioned criterions. The list includes

1. javax.servlet.Filter Package
2. JavaServerFaces (JSF) Framework
3. Struts Framework
4. Spring & Direct Web Remoting (DWR) Framework

1. javax.servlet.Filter Package

Java has the advantage of validating user input using its inherent *javax.servlet.Filter* interface package. This package intercepts incoming requests and outgoing responses to view, extract, modify and/or process data being exchanged between a client and a server [51]. More clearly, the filters allow HTTP requests to be pre-processed before it accesses a resource (either a servlet/JSP/static file) and similarly to post-process HTTP responses before returning to the client.

Java filters have a multitude of functionality, and extracted from the Servlet 2.3 Specification, some of its uses, especially in addressing security issues, are as follows:

The Java Servlet Specification version 2.3 is a publicly available document that outlines the standard for the Java servlet API.

Filter Uses	***Purpose***
Input Validation	▪ Intercepts and examines HTTP requests before invoking a resource. The examination includes enumerating and collecting all parameter values. ▪ Ability to modify request/response headers by providing customized version of the request/response that wraps the real request/response. ▪ Capable of handling multipart/form data like POST requests that handles even file uploads.
Authentication and Authorization	Check if all request and response comply with the authentication and authorization requirements of the application. For example, checks if there's a user object in the current session. If there isn't, the request is forwarded to the login page.
Logging and Auditing	Logs every request/response that takes place between the client and web server. For example, a filter known as *time filter* may be used to measure the time it takes for a request to be processed. If the time filter is the last filter in the chain, the servlet execution or page access is timed. There is also another filter called the *clickstream filter* which tracks user requests (clicks) and request sequences (clickstreams) to enable site administrators to know which sites are being visited or what pages are being accessed.
Encryption	Automatically redirects the servlets that need encryption to the HTTPS port. Enforces pages that contain private and confidential information (like passwords) to be downloaded with HTTPS. If the browser tries to load that page with HTTP, the server would *redirect* it to the equivalent HTTPS URL.

Table 3: Multiple uses of the Java Filter (javax.servlet.Filter) Package

Each of the filter functionality can be further customized or new functionalities can be created and added depending on the requirements of the application. Citing from [51], multiple filters add increasing levels of security for the application.

A running example of the use of filters is as follows; upon receiving a request from the client, the servlet container decides which filters to apply. Filters are defined and mapped to a resource (servlet/JSP/static file) in the deployment descriptor section of a web.xml file. A filter then has three options, handled by three interfaces (Filter, Filter chain, Filter config), before it calls a servlet/JSP/static file. The options are:

- Pre-process the request and send the result to the caller (*Filter*)
- Pass on the request to a resource or another filter (*Filter chain*)
- Process the request accordingly before passing it on (*Filter config*)

The filter chains are built based on the order of the filter-mapping entries in the deployment descriptor. Many filters, each performing specific tasks can be chained together. If the current request has reached the last filter in the filter chain and if all filter constraints have been checked and passed, the filter will allow the request to invoke a particular resource. The same order of steps is followed when displaying a response to the client.

Suppose a filter is used to verify the existence and validity of HTML form elements. If one or more constraints are not satisfied, the request is sent to a particular servlet (i.e. NoElements.java or InvalidEntries.java). On the other hand, if all constraints are fulfilled, the filter invokes the requested servlet (MyServlet.java). A diagram illustrating this example, excerpted from [68] is shown below

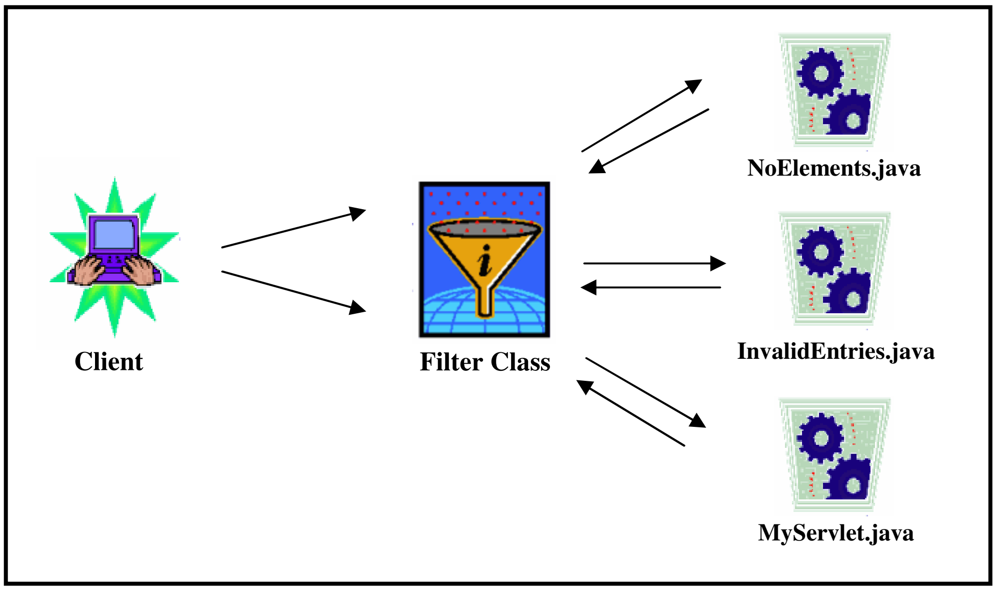


Figure 16: Input validation using the javax.servlet.Filter Package

2. *JavaServer Faces (JSF) Framework*

JavaServer Faces (JSF) offers a component based API for building rich user interfaces in Java based web applications. It comprises of many different components that are customizable and comes with additional functionalities like event handling, page navigation, input validation, data conversion and others.

From the input validation standpoint, JavaServer Faces (JSF) offer four forms of validation via its Validation API (*javax.faces.validator*) [73];

Built-in validation	Performs standard checks like data type and data range checking

Application level validation	Checks if the data complies with the application's business rules
Custom validation	Used to specify validation controls not provided in built-in validation
Backing beans validation	Implements customised validation using the backing bean method.

The backing bean method is used for application specific functions. It eliminates the need of a separate validator (for validation) and listener (for receiving/handling events) interface. The validator and listener methods are coded within the same bean that defines the properties for the components referencing these methods. These methods will access the component's data and determine how to handle the event or to perform the validation associated with the component.

The advent of the Ajax4JSF framework enables Ajax (Asynchronous JavaScript and XML) functionality to be integrated with JSF applications. Ajax enables user input in web forms to be validated near real-time. This means that while the form is still being filled, when a user proceeds to fill the next field, the previous input is asynchronously validated by a server-side component. If the data received is invalid, an error message is returned to the browser and dynamically displayed next to the invalid input entry. With Ajax4JSF, developers needn't include client-side scripting in their application. Moreover, since Ajax4JSF is an open source framework, developers have full access to the source code, which allows them to create own customised solutions.

3. *Struts Framework*

The Struts framework is used for developing large scale Java applications that uses servlets and/or JSP. Originally designed by Apache.org, one of the most powerful aspects of Struts is its support for creating and processing input from web forms using the Validator framework.

The Validator framework in Struts performs validations on form data. Typically, each Validator (e.g. creditCard, date, email) represented in a Java class, provides a single validation rule. When the framework makes a call to a Validator, the rule represented in the class is executed. More classes (also referred to as Validators) can be chained together to form more complex sets of rules.

A few of the several benefits from using the Validator framework are

- It consists of several built-in validation rules
- Server-side and client-side validation rules can be defined in one location reducing programming redundancy
- Configurations of new rules and/or changes to existing rules are simpler as all of the validation rules and details are declaratively configured in a single file
- Supports Internationalization which avoids compatibility problems
- Supports regular expressions which determine if strings of characters fit specific inputs such as an email address, postcode, telephone number, etc.

4. *Spring & Direct Web Remoting (DWR) Framework*

The Spring Framework developed by Interface21 is designed to be easily fitted into the Java EE application framework. Concerning input validation, Spring features a *Validator* and *Data Binder* interface that make up the validation package. Validation in Spring takes place at both the presentation and business logic layers. The data binder allows user input to be dynamically bound to a Java bean while the validator interface is responsible for validating application specific objects. With the data binder, developers needn't have to write additional code to bind inputs to particular objects.

Along with Direct web Remoting (DWR), Spring allows the implementation of both client-side and server-side validation. DWR is a Java open source library that dynamically generates JavaScript based on Java classes. It consists of

- A Servlet running on the server that processes clients request and returns response
- A JavaScript running in the browser that dynamically validates the webpage

DWR is responsible for binding request parameters to a Java instance. Hence, when DWR is used, the Spring framework needs only to use its *Validator* interface to take care of validation of the already bounded Java instance.

Discussion and Conclusion

The existence of the *javax.servlet.Filter* package is not widely known among Java developers. Its specific uses and implementation methods are not easily available or discussed in great lengths. The filter class does provide a lot of benefits not only with respect to validating input, but also for authentication, authorization, session management, logging, encryption and various other uses. Moreover, since filter code are written in separate classes, it becomes easier for classes to be reused and maintained. Redundant pieces of code are no longer found in several locations of the application code.

Filters are specified declaratively in a web application's deployment descriptor (web.xml file). This feature allows filters to be easily added or removed without having to alter any of the existing application code. In the same way, since each resource is mapped to particular filters in the web.xml file, the resources can also be conveniently added, removed or updated. Moreover, filters can be applied to any type of resource served by a servlet engine ranging from pure html files, URLs, graphic files, JSP pages, servlets [67].

Contrarily, defining filter tags and mapping them individually to appropriate resources in the deployment descriptor may be a cumbersome and error prone task especially for large scaled web applications. These mappings are important as they specify the filters, the affected resources and the corresponding filter chain. Therefore, this part of filter implementation requires careful concentration.

Furthermore, filters are used in a threaded environment [67]. Hence, a single instance of the filter class needs to handle one or more incoming requests/responses that relates to it. Each request will execute methods on the same filter object in different threads. Problems may arise if one thread modifies the shared object while another accesses it. The second thread may not obtain the result it expects even if it is only retrieving data from the object, hence causing an inconsistent state within the object. Therefore, programmers must be aware of this and apply thread-safe programming techniques so that the filters are able to execute safely without unwanted interaction between the threads.

One question that still remains unanswered: Does the use of filters affect the performance of web applications? Also, the use of filters to access databases or other backend or subsystems are not clearly explained. The study showed that filters are use to invoke resources like servlet/JSP/static files. These and unfamiliarity of the filter package could be some of the reasons why filters don't seem to be a common practice among Java developers.

Overall, the filter package is a good API that intercepts and processes requests before accessing resources. Nonetheless, Java developers need to add specific validation checks for completeness. The policies and guidelines developed in Chapter 4 under Input Validation (Section 4.1) may be useful.

The JavaServerFaces (JSF) framework provides a comprehensive coverage of input validation routines. The separation of code between the application logic and presentation layer enables developers to focus on specific parts in the implementation. Furthermore, validation is done in two ways; direct validation and delegated validation. In direct validation, developers embed the validation logic within the component itself and the validation becomes applicable only for that component. Delegated validation enables the validation logic to be shared across many components.

More and more developers are considering the use of or are already using JSF in developing web application. Its component based approach, its presence in Java's Enterprise Edition; along with strong ongoing community support are some of the several reasons of choosing the JSF framework. The main advantage of using the JSF framework, is the presence of event driven framework that altogether is able to handle user interface component interactions, input validation as well as page navigation and rendering.

There are no major distinctions between the Struts, Spring and JSF Frameworks. While JSF is component based, Struts and Spring consists of many short, cohesive and easy to read methods in its validation package. Moreover, both the Spring and Struts framework can be easily integrated with JSF. All three frameworks are based on the MVC (Model-View-Controller) architecture which enables security related functions like validation to be implemented on all three layers. The frameworks are also extensible allowing programmers to add customised validation checks especially for complex applications.

One main difference between the frameworks are; due to its strong developer and user community support, the Struts validator framework is tested very often giving developers more confidence on its reliability. Unfortunately, Struts is only applicable for applications that work in a web browser. The JSF and Spring framework however are compatible in both desktop and mobile browser clients. For example, the user interface components in JSF only represent the *attributes*, *behaviors* and *events* for a component and not the actual display. The manner is which input fields are displayed are handled by separate components called *Renderers* which will take care of displaying the user interface components on different client surfaces; one client may be the HTML Browser running on a PC and the other may be the WML Browser running within a mobile phone. This enables applications to be made easily available and accessible over a wide range of web enabled devices.

3.1.1.1 Buffer Overflows

Having studied the details of buffer overflows in Chapter 2, to control its occurrence, protection measures include

- Specifically reserve sufficient amounts of space for stacks/arrays/heaps
- Prevent unauthorized access or tampering on an application's memory space
- Ensure that all received input are correct and within the boundaries of defined buffer lengths

The Java language comes pre-built with measures that help in relieving buffer overflow vulnerabilities. Most of the measures are due to the property of the Java language itself. The measures include:

1. Type Safety
2. Class Loader and Byte-code Verifier
3. Bound Checking Mechanism
4. Garbage Collection System

1. Type Safety

The Java language is designed to enforce type safety which is one of the essential elements of Java's security features. With type safety, programs can be controlled from unauthorized memory accesses, preventing buffer overflows and data access violations. Each Java object has a class which resides in some part of a program's memory. Each class in turn defines both a set of objects and operations to be performed on the objects of that class. Only certain operations are allowed to manipulate objects of that class [44]. Type safety ensures that an object's method may not be executed unless the operation is valid for that object.

Java's Virtual Machine (JVM) labels every object in memory with a class tag. One way in which Java enforces type safety is by checking the class tag of an object before invoking any operation on the object [44]. This approach is known as *dynamic type checking* which executes at runtime. Another method of type checking in Java is known as *static type checking* which checks all possible executions flows of a program, at compile time. Therefore, all expressions, arguments to functions and variables must be initialized with a type. To improve the program's performance it is best to use *static type checking* whenever possible. In using the *static type checking* option, Java checks the program before it is run and conscientiously determines if a particular tag checking operation will always succeed. Once determined, then the check can be safely removed significantly increasing the speed of the program. On the other hand, if there is a mistake in the type checking procedure, then the call is detected before it is executed.

The type safety feature of the language is capable of validating memory allocation and memory access at runtime, protecting against stack overflows. Type safety also guarantees a program against events such as treating pointers as integers or vice versa, and sign-conversion bugs [47].

2. Class Loader and Byte-code Verifier

Malicious byte-code can be created by hostile compilers or compiled from languages like C++ into Java byte-code to violate the rules of the Java Virtual Machine (JVM). To avoid this possibility, the byte-code verifier in Java verifies if the incoming streams of byte-code can be "trusted" by the JVM. The byte-code verifier ensures that the code passed to the Java interpreter, in the JVM can be safely executed without breaking the Java interpreter.

The verification ensures that the code doesn't contain falsified pointers, operand stack overflows or underflows, doesn't violate access restrictions and only accesses objects as what they are. For example, *InputStream* objects are always used as *InputStreams* and nothing else. Furthermore, object field accesses should be in either one of the three legal categories, i.e. private, public, or protected.

However, as mentioned in [75], byte-code verification by itself does not guarantee secure execution of the code. Bounds, null pointer and access control checks must still be carried out.

The Java class loader is responsible for loading Java byte-code into the JVM and then converting the raw data of a class into an internal data structure representing the class. Each loaded class are treated as distinct types by the JVM and are each associated with a unique identifier by the JVM. Hence, once a class is loaded into a JVM, the same class will not be loaded again.

The class loader works alongside the Java security manager and the access controller in the JVM. It's responsible for locating and fetching class files, consulting the security policy, and associating the class object with the appropriate permissions. The security manager depends on class loaders to correctly label code as trusted or untrusted.

Class loader contributes to security in three ways

- Prevents malicious code from interfering with trusted code by providing separate namespaces for classes loaded by different class loaders. No two classes with the same name can exist in the same namespace.
- Enables trusted packages to be loaded with different class loaders than untrusted packages.
- Places code into protection domains that will determine which actions/methods the code is allowed to invoke.

3. Bound checking mechanism

The Java specifications require that exceptions be raised for any array access in which the array index expression evaluates to an index out of bounds. This built in bound checking mechanism in the JVM helps to prevent buffer overflow vulnerabilities. Each method's code in Java is independently verified and validated, instruction by instruction. For example, the dataflow analysis performed during verification would make sure that if an instruction uses operands (from the operand stack), there are enough operands on the stack, and that they are of the proper types. The verifier also ensures that code does not end abruptly with respect to execution. This significantly reduces the risks of buffer overflows, sign-conversion bugs, and integer overflows in applications.

However, there is a drawback especially for high-performance parallel computing. In practice, array-bounds checking in scientific applications may increase execution time by more than a factor of 2 [46].

4. Garbage collection system

The garbage collection system in Java helps to prevent exploits relating to overloaded memory. Java has built-in ability to automatically clear used memory with its garbage collection system. The memory allocation and destruction is managed by the Java runtime execution engine in the JVM. Should an application require more interaction with Java's garbage collection system (e.g. when programming memory sensitive caches), classes from the *java.lang.ref* can be used.

Discussion and Conclusion

Java features prominent properties that aid in reducing buffer overflow vulnerabilities. It implements type safety, prevents malicious byte-code, offers bounds and access control checks and a garbage collection system, for safe and clean coding. However, all of these properties are only as secure as its JVM. Hence, developers need to ensure that the JVM is constantly patched or updated with newer or improved releases.

One drawback with regards to its garbage collection mechanisms is that memory is only cleared once the application closes. Hence it's important that for a large or critical application, memory consumption is constantly monitored and that applications instruct the garbage collector to free memory at certain time intervals.

More explicit coding policies to avoid buffer overflows vulnerabilities can be found in Section 4.1.1.

3.1.1.2 Cross Site Scripting (XSS)

Chapter 2 showed that XSS vulnerabilities may be prevented by

- Setting the character set and language locale for each page generated by the web server
- Filtering and validating all user input
- Encoding all special characters with its equivalent ASCII/UNICODE/ISO format
- Filtering and encoding output of all data types especially for special characters

Section 3.1.1 of this chapter lists the input validation techniques in Java that can be used to prevent XSS as well. This section studies the encoding measures to help prevent malicious inputs/scripts/commands from being stored, displayed or executed.

Modern web applications cater for a wide range of languages (English, Chinese, Indian) which deals with more than just ASCII characters. Because of the different possible representations, it is important to ensure the use of appropriate encoding mechanisms. The process of converting data into a standard form is known as canonicalization. This process is important to understand especially in terms of input validation because if canonicalization is not performed, data can be misinterpreted and validation may miss an attack.

Canonicalization in web applications is the encoding of characters into its respective HTML equivalent. One common representation of characters used on the web is the Unicode UTF-8 representation, which is also the default output encoding scheme. Besides converting data to a standard form, it translates a 31-bit character set into an 8-bit representation to reduce the number of bytes transferred between computers.

To handle special kinds of input characters (sometimes referred to as meta-characters like & < > ! $) that may have special meanings in certain context; encoding mechanisms are encouraged as a first line of defence in preventing malicious hyperlinks, commands and/or scriptlets from being sent to the web server or displayed in the browser. Character encoding in web based applications can be divided into URL encoding and HTML output encoding.

URL encoding is the process of converting strings (e.g. from forms) into valid URL format which can be safely transmitted across the Internet. URL encoding is sometimes also termed as percent-encoding. It is also used in the preparation of data in the "application/x-www-form-urlencoded" media type, as seen in email messages and the submission of HTML form data in HTTP requests.

HTML output encoding, on the other hand is used to represent HTML reserved characters in its reserved format (i.e. for the texts inserted between the tag <b>…</b> to appear bold). HTML has several reserved characters which are used to create HTML pages. Popular examples of HTML reserved characters are <, > and &. The resulting output encoding for these characters are

Reserved Characters	HTML character entities
<	&lt
>	&gt
&	&amp

Table 4: Special characters encoded as HTML character entities

These characters must be presented to users in the form of HTML character entities where each entity is composed of an ampersand, a short-hand name for the character, and a semi-colon.

Java provides the following two classes for encoding purposes;

1. java.net.URLEncoder
2. StringEscapeUtils

1. java.net.URLEncoder

This class contains methods to perform both URL encoding and decoding. The purpose of URL encoding is to allow non-URL compatible characters to be passed via the URL. According to RFC 3986, the characters in a URL have to be from a defined set of reserved and unreserved ASCII characters. Any other characters are not allowed. When non-alphanumeric characters from the reserved set have a special meaning in certain context, then the character must be URLEncoded.

The use of an Internet media type (originally called MIME type and sometimes Content-type), specified in the header of HTTP protocols, is to allow the use of a variety of formats for web forms. The default encoding scheme for all forms is ‘application/x-www-form-urlencoded’. When encoding, the following rules provided by *java.net.URLEncoder* applies:

- The alphanumeric characters a-z, A-Z and 0-9 remain the same.
- The special characters . - * _ remain the same.
- The space character is converted into a plus sign "+" or its equivalent %20.
- Fields with null values should be omitted.
- All other characters are considered unsafe and are converted into one or more bytes using some encoding scheme. Examples of encoding schemes include ISO-8859-1 (Latin-1), UTF-8 and ASCII. The default encoding scheme is UTF-8.

URL decoding is used to decode the encoded URL.

2. *StringEscapeUtils*

Special characters like < > " ' \ % ; & $? # : = , ~) (+ - enable scripts to be executed, hence contributing to XSS attacks. The *StringEscapeUtils* class provides a set of encoding functions that escapes and unescapes string literals for Java, JavaScript, HTML, XML, and SQL. Escape and unescape characters are used to format and at the same time prevent special characters from causing interpretation errors or malicious executions.

An example showing the use of this class is as follows:

- escapeJava - Escapes the characters of a string using the Java string rules.
  - input string: He didn't say, "Stop!"
  - output string: He didn't say, \"Stop!\"
- unescapeJava - Unescapes any Java literals found in a string.
  - For example, it will turn a sequence of '\' and 'n' into a newline character, unless the '\' is preceded by another '\' (i.e. '\\').

Discussion and Conclusion

The standard coding mechanisms mentioned above are sufficient as countermeasures for controlling obvious XSS vulnerabilities. However, even with the availabilities of such packages, one of the reasons why XSS vulnerabilities still exist in web applications may be due to the fact that encoding and decoding actions are scattered in various sections of the application code. Furthermore, the order in which characters are encoded and decoded are not given enough attention.

For example, a web server may initially decode all URLEncoded characters and perform the necessary verification checks (i.e. monitoring all "\..\"paths that it does not expand beyond its defined limit). But, what happens if the contents of the URL had been encoded multiple times? The decoding and verification checks may be circumvented. Moreover, if the information is passed to other components of the application, it may go through additional decoding, and result in an unexpected situation.

Besides the above mentioned factors, encoding input before storing in a database also pose a problem. Encoding expands data strings, hence affecting the restricted length of database fields. Therefore, encoding should be used to check if the input data contains malicious or special characters. All input data should be decoded before being stored and encoded again before displaying on the browser. Hence, encoding should not only be performed on user input but also when reading data from a file, database or any other external sources.

Please refer to Section 4.1.2 for more coding policies on avoiding XSS vulnerabilities.

3.1.1.3 Injection Flaws

The previous sections discussed the validation and encoding options provided in Java that if implemented may help to avoid some of the web input vulnerabilities. Similarly, vulnerabilities related to injections flaws can be prevented by

- Ensuring that only filtered and validated inputs are sent to the web server
- Including validation methods like data conversion and regular expressions

Special characters, malicious commands, or command modifiers are prevented from bypassing the web application by incorporating the techniques discussed in the above categories. Additionally, regular expressions are effective in determining if the input text from web forms, URL/query strings, cookies, meets the expected input requirements. It is a mechanism to specify a textual pattern and verifies the presence of the pattern in a given input field. Hence, irregular formats or input not conforming to the desired patterns are not further processed.

When a Java program receives user input, it may occasionally need to be converted from one form (e.g., string) into another (e.g., double or int) for processing. This conversion can be handled using one of the input validation techniques presented in Section 3.1.1.

Java offers regular expressions mechanism through its *java.util.regex* package.

1. java.util.regex Package

Java's regular expression package can be applied in a wide variety of applications. The package consists of the *java.util.regex.Pattern* class, the *java.util.regex.Matcher* class and an exception class *java.util.regex.PatternSyntaxException* that altogether represents the regular expressions framework in Java.

A regular expressions pattern is typically specified as a combination of two types of characters, *literals* and *meta-characters*. Literals are normal text characters (a, b, c, 1, 2) while meta-characters (e.g. * ' $ \) convey special meanings to the regular expression engine. The *'Pattern'* class is the compiled representation of the specified regular expression string. The *'Matcher'* object does the matching operations on specified character sequences and provides additional functions to access and use the results from the match.

This package can be used to validate URLs, passwords, emails, web addresses, perform text conversion and various others. For instance, to validate a URL, the URL pattern matches the hostname followed by optional path names as advocated by the regular expression.

Discussion and Conclusion

The use of regular expressions exerts stricter rules for validating certain input fields. It is an added form of validation for inputs that require specific patterns which would otherwise require tedious coding rules. Some of the frameworks discussed under the Input Validation category, like Struts, include the use of regular expression in its framework. Incorporating regular expression in the other frameworks is also not hard. Therefore, the use of regular expressions to validate input should be enforced in all web applications.

However, using regular expressions for every input may slow down an application especially if the input string is long or involves complex expression rules. A solution may be to pre-compile regular expressions into assembly which would allow the expression to run faster and also limit its use to specific fields.

Another drawback with the use of regular expressions is that it may have a slightly different meaning across languages that also support regular expressions (like Perl, .NET, etc.). Therefore regular expressions may not be portable across languages.

To prevent against injection flaws a combination of measures is required. Besides input validation, character encoding and regular expressions, it is imperative to include mechanisms like encryption. Encryption options in Java are discussed in Section 3.1.4. Moreover, developers need to identify all input entry points in the application and understand the affects of different special characters when accessing or invoking external applications.

3.1.1.3.1 SQL Injection

To thwart SQL Injection vulnerabilities, it is vital to apply controls specific to the SQL language, to protect the integrity of databases. This form of validation prevents attackers from directly accessing an application's database through malicious concatenated SQL statements.

An attacker can inject a variety of characters that are then translated by the database into executable commands. These characters may vary depending on the database used but often include a variety of symbols like + - , ‘ “ _ * ; | ? & = as well as reserved words in SQL such as DROP, OR, UNION, JOIN, etc. Therefore, it is important that inputs used to access the database are not parsed literally.

There exist two general methods to control against SQL injection attacks;

- Use prepared statements with parameterized SQL when querying a database
  Prepared statements are defined once in the application and are executed as many times as required with different parameters. These parameters hold the value of the user input. This together with strong input validation implementation protects an application from SQL injection attacks.

  A typical prepared statement in Java looks like the following:

  ```
  String selectStatement = "select * from Country where code = ? ";
  PreparedStatement prepStmt = con.prepareStatement(selectStatement);
  prepStmt.setString(1, userId);
  ResultSet rs = prepStmt.executeQuery();
  ```

  The “?” placeholder is eventually replaced with appropriate input values.

- Use stored procedures with parameterized SQL when querying a database
  Similar to prepared statements are stored procedures which are groups of SQL statements that reside in the database server. Stored procedures can be used over the network by several clients using different input data while prepared statements are associated with a single database connection. This is one of the features that differentiate stored procedures from prepared statements. Moreover, stored procedures is said to considerably reduce network traffic and improve the application’s performance.

  SQL statements in stored procedure are saved separately depending on their function. For instance, the statement *SELECT * FROM Country* is stored in a file named sp_displayallcountries. The command *execute sp_displayallcountries* tells the database server to execute that particular stored procedure.

The main distinction between stored procedures and prepared statements are that stored procedures are stored in the database server while prepared statements are stored within the application server.

In Java, the following three classes cater for the use of both prepared statements and stored procedures;

1. *java.sql.PreparedStatement*
2. *java.sql.statement*
3. *CallableStatement*

The Java Database Connectivity (JDBC), which is the standard API for Java applications accessing database data, is able to cater for both prepared statements and stored procedures. For prepared statements, the JDBC driver uses the *java.sql.statement* and *java.sql.PreparedStatement* classes to pass SQL strings to the database for execution and to retrieve any results. The variables or user input are encapsulated and special characters within them are automatically escaped before being handled by the target database.

Similarly, the JDBC allows a call to stored procedures via the *CallableStatement* class. This class is a subclass of the *java.sql.PreparedStatement.* Stored procedures accept data (i.e. input parameters) at execution time. Successful or failed queries are signalled with exceptions.

Discussion and Conclusion

Prepared statements use bind variables to prevent input variables from being passed directly into the prepared statement. Bind variables are substitution variables that are used to replace the "?" placeholders. The JDBC has built-in support for bind variables.

Prepared statements maintain a connection to the database only for the duration of a request. The prepared statements would need to be recompiled every time the request is executed. The solution is to use a cache inside the Java EE server. This server keeps a list of prepared statements for each database connection in the connection pool manager. When a prepared statement is invoked, the server checks if the statement had been previously compiled. If it was, the prepared statement object will execute before returning the result to the application.

Different from prepared statements, stored procedures are compiled before user input is added. Optionally, owners of particular stored procedures can grant users permission to execute a stored procedure independently of underlying permissions.

One problem with both these methods, since JDBC automatically escapes special characters like single quote, how would the application handle names that include single quotes, i.e. O'Connor, O'Reilly. One possible solution is to use single quotes for encapsulating data. Hence, if a text field receives input O'Connor; the string is split on all single quotes and joined together using a backslash-escaped single quote. Therefore, the O'Connor input would be processed as 'O'\''Connor'.

However, it is important that there exist no space between the words 'O' and 'Connor'. There could be other approaches to handle this kind of input though.

From a performance perspective, stored procedures help to reduce latency in applications as its execution takes place in the database server. It also reduces the number of network trips which has a dramatic effect on performance. Stored procedures also provides much flexibility in migrating applications for instance from ASP to J2EE. Changes need only be made at the application layer as much of the business logic will remain in the database. Unfortunately, not all databases support stored procedures. Therefore, depending on the choice of DBMS portability may not be an issue.

3.1.2 Authentication and Authorization

Recapping from Chapter 2; *authentication* refers to the process of identifying a user usually based on their username and password while *authorization* determines the level of access the authenticated user may have to the protected contents and resources of an application. Authentication and authorization are important to ensure that protected contents and resources are only accessible to those identified with the right permissions.

Below describes the Java Authentication and Authorization Services (JAAS) package in Java EE that helps in performing authentication and authorization mechanism for Java applications.

1. Java Authentication and Authorization Services (JAAS)

According to Sun Microsystems, JAAS consists of a set of APIs that may be used for

- *Authentication:* determining the identity of an entity (user/system/process) attempting to execute parts of a Java code (i.e. stand-alone Java technology-based application, an applet, an Enterprise JavaBean (EJB) component, or a servlet)
- *Authorization:* ensure that only entities with right permissions are able to perform certain operations

It also includes the Java implementation of the standard Pluggable Authentication Module (PAM) which allows the use of multiple authentication technologies like UNIX, Kerberos, RSA, smart cards or authentication approaches such as login, passwd, rlogin, telnet, ftp, to be added or removed without affecting any parts of the application code.

Let's first take a look at the authentication mechanisms for JAAS.

Authentication
Table 3 shows a brief study of the core classes in the authentication process of JAAS. Contents from this table are summarised from the JAAS Reference Guide for the Java SE Development Kit 6.

Class	***Description***
Subject *javax.security.auth.Subject*	• Represents the source of a request (i.e. user/system/process) • Usually associated with Principals, public credentials (public key certificates), private credentials (private crypto keys)
Principals *java.security.Principal* and *java.io.Serializable* interface	• Each subject is uniquely associated with one or more principals. For example like name, Social Security Number (SSN), organization, login id that distinguishes it from other subjects. • Subjects may potentially have multiple identities. Each identity is represented as a Principal within the Subject. Principals simply bind names to a Subject.
Credentials Any class can represent a credential. There are two interfaces related to credentials: - *javax.security.auth.Refreshable* - *javax.security.auth.Destroyable*	Security related attributes that require special protection, such as private cryptographic keys, are stored within a private credential set. Credentials intended to be shared, such as public key certificates or Kerberos server tickets are stored within a public credential set. Different permissions are required to access and modify the different credential sets. The *Refreshable* interface enables a credential to be restricted to a lifespan, requiring the credential to refresh itself. The *Destroyable* interface on the other hand, provides the capability of destroying the contents of a credential.
LoginContext *javax.security.auth.login.LoginCont ext*	• Describes the methods used to authenticate Subjects • A separate LoginContext is used to authenticate each different Subject.
Configuration *javax.security.auth.login.Configurat ion*	Specifies the authentication technology/login module to be used. Applications can use more than one login module/authentication technology at any one time. For example, one could configure both a Kerberos LoginModule and a smart card LoginModule.
LoginModule *javax.security.auth.spi Interface LoginModule*	Executes the authentication modules/technologies implemented in the application. For instance, • A module to perform username/password authentication • Interfaces to hardware devices such as smart cards or biometric devices.
CallbackHandler *javax.security.auth.callback.Callba ckHandler*	• Interface between the LoginModule and the LoginContext which obtain authentication information from the user. • Gather inputs such as a password/pin number or supply information to user's (i.e. status information)

Table 5: Core classes of Java's JAAS Authentication

The Subject, Principals and Credential classes are categorised as common classes while the LoginContext, LoginModule, Configuration, and CallbackHandler are set as authentication classes and interfaces.

Generally, an authentication process using JAAS first instantiates the *LoginContext* object. The parameters from *LoginContext* are used as the index in *Configuration* to discover the authentication technologies or modules that should be used for authentication. The *LoginContext* then invokes the login method which unloads and later executes all of the *LoginModules* for authentication. If the login method returns without throwing an exception, then the overall authentication succeeded. To logout the *Subject*, the logout method must be called. As with the login method, the logout method invokes all of the logout method for the configured modules. The *LoginContext* is responsible in returning the authentication status to the application.

Next, we take a look at the authorization mechanism in JAAS.

Authorization

Once authentication successfully completes, the JAAS authorization component is invoked. This component makes sure that authenticated Principals have the necessary access control rights (permissions) required to perform security sensitive operations in the application. The authorization component associates the Subject with an appropriate access control context. Hence, when a Subject attempts an access to a file or contents in a database, the Java runtime consults the policy file to determine which Principal(s) may perform the operation. If the access does not contain the designated Principal, an exception is thrown.

The classes involved in JAAS Authorization are the Policy, AuthPermission and PrivateCredentialPermission classes.

Class	***Description***
Policy *java.security.Policy*	• Documents the access control policy for the entire application. • Specifies or identifies the privileges assigned to Principals attempting to execute certain operations.
AuthPermission *javax.security.auth.AuthPermission*	Used to guard access to the Policy, Subject, LoginContext and Configuration objects.
PrivateCredentialPermission *javax.security.auth.PrivateCredentialPermission*	Protects access to a Subject's private credentials. The Subject is represented by a set of Principals.

Table 6: Core classes of Java's JAAS Authorization

Discussion and Conclusion

JAAS is now a popularly growing standard for enforcing authentication and authorization controls in Java applications. Most Java applications requiring a strong authentication and authorization mechanism recommends the use of JAAS. It has since become a generally accepted technique.

JAAS authentication is performed in a pluggable fashion. This permits new or updated technologies to be plugged in without requiring any modifications to the application code. Another advantage of JAAS is its ability to perform multiple authentications. Moreover, JAAS provides both code-centric and user-centric access controls where permissions are not granted based only on what code is running but also on who is running it.

However, notably, as also featured in OWASP, JAAS requires additional development efforts before it can be used for access control in web applications. For instance, JAAS lacks features like automatic timeout mechanisms. Furthermore, there are a few areas that remain a bit vague

- Can the authentication and authorization be extended to access multiple servers (i.e. database servers, other application servers, etc)
- The single-sign on solution requires the use of compliant Java EE versions, and hence many not be practical for use in applications that integrates with applications of different platforms
- How are the credentials managed in JAAS?

Besides JAAS, another method for authentication and authorization in Java is using servlets. Unlike JAAS, the servlet's security model is intended specifically to handle the requirements of web applications. The modules include user authentication/authorization by servlets, user authentication for single sign-on, fetching client certificate and security for SHTML and CGI. Details of these modules are not discussed in this thesis.

3.1.3 Error Handling and Logging

A good error handling process is able to anticipate, detect, and resolve errors and exceptions that occur in an application, expected or unexpected. It is important that error details are appropriately formulated before revealing messages to end users. Sensitive information gives great insights into the inner workings of an application.

There are three rules that must be kept in mind when handling exceptions. The rules are
(1) Be Specific – get as explicit as possible with the types of exceptions to be handled
(2) Throw Early – throw an exception as soon as the application detects an erroneous piece of information or unexpected attempts
(3) Catch Late – don’t catch an exception that doesn’t have any recovery clauses. Redirect the exception to a generic page and try to recover the application

Java has three main classes to handle exceptions and errors in applications. There are

1. java.lang.Throwable,
2. java.lang.Exception, and
3. java.lang.Error,

As for logging purposes, the following class can be utilized

4. java.util.logging

1. java.lang.Throwable

Before an exception is caught, the affected Java code must throw one. The *java.lang.Throwable* class is the superclass of all errors and exceptions in the Java language. This class holds two important subclasses namely the Error (*java.lang.Error*) and Exception (*java.lang.Exception*) class.

2. java.lang.Exception

This Exception class indicates abnormal conditions that a reasonable application might want to catch. Exceptions are derived from the Throwable class or one of its subclasses and are thrown to signal abnormal conditions. If exceptions are not caught, it could result in a dead thread.

The exception class is built of the try, catch and finally blocks.The try block contains the code that might throw an exception while the catch block contains the code that handles the thrown exception. The finally block is used to clean up any resources created in the try/catch block, and should be executed whether an exception is thrown or not.

Java also allows its developers to create customized exception classes using the *java.lang.Exception* class. This is done when the exception type needed is not currently available in the Java platform or to represent specific problems that can occur within the classes they write or to differentiate errors between their code and errors that occur in the Java development environment or any other packages.

3. java.lang.Error

The error class indicates serious problems that an application may not be able to handle. Most such errors are abnormal conditions like 'out of memory'. To distinguish between errors and exceptions, written code should throw only exceptions, while errors are usually thrown by the methods of the Java API, or by the Java Virtual Machine itself.

2. java.util.logging

This logging API can be used to capture information related to security failures, configuration errors, performance bottlenecks, and/or bugs in the application or platform. The two most important classes in this API are the Logger and Handler class. Both the classes are responsible for logging messages to a defined location.

Discussion and Conclusion

The prevention mechanisms presented in this category are well known to Java programmers and can be found in many Java textbooks or tutorials as the basics for handling errors and exceptions in Java. The most important aspect in this category is to limit the amount of information relayed to end users. Formulating appropriate error messages are often under-exercised by application designers and developers. More often, error messages are constructed to help developers debug and modify the application.

Since the task of exhausting all possible errors or exceptions is unrealistic, a more careful implementation, which does not reveal too much information, should be used. For faults that are caused by user input and to which the application is able to detect, the application should return a message that is meaningful but also succinct. For unanticipated errors or exceptions that is caused by the operating system, compiler, server, application or database; revealing too much or unnecessary information may pose a threat.

As specified in Section 4.4, all messages should be attached with a unique identifier (i.e. JK01). These identifiers are documented in a separate message handbook. In this way, developers can refer to both the application logs and the handbook to obtain detailed information about the errors or exceptions and how to proceed fixing. At the same time, messages sent to the user are filtered and sufficiently understandable.

With respect to the logging class, developers should make sure that the log files are kept in restricted locations and all sensitive information within it are encrypted. More policies and guidelines for logging web applications can be found in Section 4.4.

3.1.4 Insecure Storage

In any application, it is of paramount importance to have a secure storage system, whether on disk or in memory, especially when handling confidential (private) user information. Most contemporary web applications collect and store information such as passwords, social security, credit card numbers and various other proprietary information. The collected information must be kept in a highly secured storage area.

Some reasons for insecure storage arise because of

- Failure to encrypt critical data
- Use of weak cryptographic algorithms, protocols or systems

- Application do not use thorough logout mechanisms that removes storage of critical data
- Insecure storage of keys, certificates, and passwords

The following are some measures in Java to circumvent insecure storage. The measures include

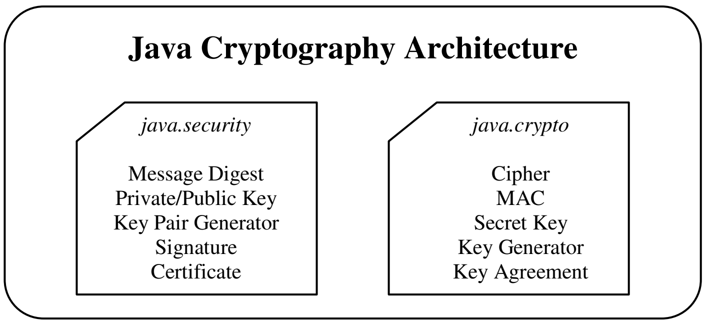
1. Java Cryptography Architecture (JCA)
2. Java Cryptography Extension (JCE)
3. Java KeyStore

1. Java Cryptography Architecture (JCA)

This package which forms part of the SDK and JCE framework provides basic cryptographic services and algorithms, which includes support for digital signatures, message digests, ciphers, key generators and key factories. JCA ensures interoperability by providing standardized sets of APIs, which implements the cryptographic algorithms and services. It implements the *java.security* and *javax.crypto* packages. JCA is also able to integrate encryption technologies for hardware devices.

2. Java Cryptography Extension (JCE)

The JCE (Java Cryptography Extension) package provides standard well known algorithms for encryption, key generation and agreement as well as Message Authentication Code (MAC) used to validate information transmitted between parties. JCE provides encryption support for symmetric, asymmetric, block, and stream ciphers. Encryption classes in JCE are found in the *javax.crypto* package which provides convenient ways to implement one of the many popular cryptographic algorithms including RSA, AES, 3DES, HMAC-MD5, HMAC-SHA1. The packages also include classes that support the storage and retrieval of encryption keys.



3. Java KeyStore

This is a protected database that stores keys and trusted certificate entries for those keys. A KeyStore stores all the certificate information related to verifying and proving an identity of a person or an application. It contains a private key and a chain of certificates that allows authentication with corresponding public keys. All stored key entries can be further protected with passwords.

Discussion and Conclusion

Java provides a comprehensive storage solution that covers numerous types of encryption algorithms and services. Moreover, mechanisms like message digests yields an easy and secure way for user authentication.

JCE can be used on any platform (Windows, Linux, and Solaris) that has a stable JVM. Because of this, applications using JCE can be implemented across platforms. Furthermore, JCE can also be extended to incorporate hardware cryptography. One major drawback, however, according to the JCE specification, is that the use of JCE is limited by export restrictions of the U.S. government. Many of the popular algorithms it contains are patented, which also restricts its usage. Fortunately, JCE extends JCA which contains all algorithms not subject to export restrictions, e.g. message digests, digital signatures or certificates.

Algorithm independence and extensibility and implementation independence and interoperability are the main advantages in JCA. When complete algorithm-independence is not possible, the JCA provides standardized, algorithm-specific APIs. In the same way, when implementation-independence is not desirable, developers can opt for a more specific implementation. This allows multiple providers providing multiple encryption technologies to integrate with each other through a framework of common classes, interfaces and methods. For example, by using the same algorithms, a key generated by one provider can be used by another provider; likewise, a digital signature generated by one provider can be verified using another provider. Most Java applications that use the JCA package have proven security functionality in many situations [79].

Key handling does not pose any threat in Java. In fact, Java provides acceptable mechanisms to share key information with another platform or programming environment, and capably handles tasks to create, read, re-create and store the keys. Java KeyStore is one example of a database that manages a repository of keys and certificates.

When using symmetric encryption, problems may arise with the use of shared keys. Of course, the solution would then be to use asymmetric encryption. In asymmetric encryption, keys used to encrypt a message are public and the private key is not transmitted over the network. However, not many asymmetric algorithms exist. Popular ones are RSA and ElGamal. Moreover, asymmetric encryption is found to be extremely slow compared to symmetric encryption. To overcome the deficiencies of both asymmetric and symmetric encryption, hybrid encryption was designed. In this technique, messages are encrypted using symmetric algorithm. The resulting private key is then encrypted using an asymmetric algorithm. Hence, the issue of transferring private keys over a separate channel is eliminated and the speed of symmetric encryption is achieved. However, hybrid encryption was not mentioned in the research on JCA and JCE.

JCE and JCA together provide the ability to implement cryptographic services with minimal knowledge of how the various algorithms work. Hence, depending on the functionality of the application, developers would need to choose the most suitable cryptographic algorithm and service from the JCA or JCE package.

3.1.5 Application Denial of Service

As discussed in Chapter 2, application DoS causes web applications to fail by causing the application to shut down unintentionally or by consuming the available resources or causing the application to hang so that legitimate users can no longer access the application.

There are no specific APIs/frameworks in Java to prevent application DoS attacks. However, some suggested measures include

- Thoroughly filtering and validating all input received from the client
- Disallow executable, operating systems or JVM specific commands from unauthorized users
    - Prevent uses of commands like *System.exit()* which forces the termination of all threads in the JVM and causes the application to shut down
- Specify certain server configuration settings
    - WebLogic has settings for MAX POST SIZE, POST TIME OUT, HTTP and HTTPS Duration
- User Windowing techniques to ensure that the server will only process a specified number of requests per unit of time. If more requests are received, the application will return an error message

Discussion and Conclusion

Today’s web applications are mostly large and complex presenting a broad area for application DoS attacks. All of the above mentioned measures does not prevent this attack altogether but at least strengthens the server to some extent. Extra configuration in firewalls, routers, network monitoring devices and intrusion detection systems is necessary to further secure the associated servers. A general rule would be to define the expected traffic and if the traffic significantly increases over a specified threshold; log the time, IP address, type of request, and any other information deemed useful.

3.1.6 Configuration Management

In this category, what’s essential is a controlled configuration management framework for managing all types of servers associated with a web application (i.e. web servers, application servers and database servers). These servers contain important files and data that are used to generate the contents of a web application, determine the process/workflow concerning the application development and management lifecycle, as well as logging communication and problem tracking matters. Some established configuration management tools specifically for addressing security vulnerabilities are provided by vendors like Altiris, BindView and LANDesk.

Apart from the tools provided by various vendors, the *SVNKit* in Java helps to defend security breaches related to configuration management activities. SVNKit is a new name for the Java Subversion library formerly known as JavaSVN. The SVNKit is used to access or modify Subversion repository from a Java application. Subversion is a version control system which is a part of configuration management activities that caters for data management and data tracking requirements in the form of a tree called a repository [37]. It does not require any additional configuration or native binaries to work on any OS that runs Java.

Some examples of projects that use the SVNKit[5] have proven results for

- Improved performance and usability
- Folders and files content browsing
- Revision details, revisions compare
- Create/delete/modify files/folders
- Multi-repository support

Discussion and Conclusion

There are a myriad of configuration management tools available and are widely used in the market today. The choice of a tool is dependent on the application as well as the process and practices of the organization building and maintaining the application. It is paramount to ensure good procedures for documenting information like what was developed, who developed it, when it was changed, why it was changed, and who authorized the change. Effective security configuration management as well as regulatory compliance assessments can help firms manage security proactively.

To attain secure applications developers must know and understand their application or at least the function they develop in and out. What are its pieces? How are they organized and related to other components in the application are some of the questions developers should answer. Then, by combining elements of vulnerability assessment, patch management, automated remediation, and configuration compliance, application risks can be considerably reduced

[5] Extracted from http://svnkit.com/

Summary on Java

Java provides a wide variety of APIs, tools, and frameworks to address each of the web application threats and vulnerabilities discussed in Chapter 2. Briefly, for input validation we have seen and discussed about the *javax.servlet.Filter* package, the *JavaServerFaces (JSF)* framework, the *Struts* framework and the *Spring & Direct Web Remoting (DWR)* framework; cross site scripting vulnerabilities can be handled using the *java.net.URLEncoder* and *StringEscapeUtils* APIs; in addition to the frameworks for input validation, injection flaws can be avoided by using the *java.util.regex* class; in the same way SQL injection is prevented with the use of the *java.sql.PreparedStatement, java.sql.statement* and the *CallableStatement* classes; authentication and authorization issues can be managed under the *Java Authentication and Authorization Services (JAAS)* package; error handling and logging should use the *java.lang.Throwable*, *java.lang.Exception*, *java.lang.Error*, and *java.util.logging* classes; insecure storage issues can make use of the various cryptographic solutions namely the *Java Cryptography Architecture (JCA)*, *Java Cryptography Extension (JCE)* and *Java KeyStore* packages; configuration management problems can be taken care by using Java's *SVNKit*. The language properties of Java provide built-in mechanisms to prevent the occurrences of buffer overflow vulnerabilities. There are no specific APIs/frameworks to prevent application DoS attacks. Most application DoS attacks can be controlled by implementing the preventive measures of all other vulnerabilities.

Even though a large collection of APIs, tools and frameworks are available, most of them are not used and when they are used their entire functionality is not exploited. Some of the reasons why Java developers do not make use of the many existing APIs, tools and frameworks is because of difficulty in finding the appropriate information. This happens because the information are poorly organized, cross references are weak or link to other documents and the lack consistency in the presentation of similar information.

To optimize the use of the APIs, tools and frameworks, Java requires its developers to posses a steep learning curve which may be difficult for entry-level programmers. Developers need to have a good understanding of

– Core class libraries (collections, serialization, streams, multithreading, localization),
– Servlets, JSP, EJB,
– Web frameworks, like JSF, Struts,
– The JVM and Java sandbox security model (class loaders, byte-code verifier, garbage collector),
– APIs like JAAS, JCA, JSE, and more

Once developers acquire sufficient amount of knowledge, it is important that they keep up-to-date with information related to these APIs, tools and frameworks.

3.2 .NET(ASP)

In close competition with Sun's Java is Microsoft's .NET technology. Both technologies are striving towards establishing a robust platform that is able to deliver secure web applications. Similar to Java, the .NET technology offers a development framework that allows the integration of different programming languages and libraries. In relation with web applications, ASP.NET which is a major part of the .NET framework creates an environment to build, deploy and manage Windows based web applications that can securely network with other web applications. It provides for improved ease-of-use, reliability, scalability and most importantly addresses certain security concerns, specifically described within this chapter.

ASP.NET, the next generation of Microsoft's Active Server Pages (ASP) technology, consists of a set of application development technologies that enables the building of dynamic web applications, including XML based web services. The following subsections discuss the various libraries, classes, frameworks and other components in .NET(ASP) that render secure web applications.

Each subsection ends with a ***Discussion and Conclusion*** box where we discuss the strength and weaknesses of the available mechanisms. This will hopefully result in more value of the information given in practice.

3.2.1 Input Validation

Referring to the input validation criterions listed in Section 3.1.1, the following subsection explains in brief the *Validation Web* controls in ASP.NET that exist to achieve the criterions for input validation.

1. Validation Web controls

Validation Web controls are controls specifically designed for performing input validation on web forms. If the user's input does not conform to any one of the validation checks, an error message is displayed to the user. All validation web controls include an ErrorMessage property that allows developers to customize unique error texts whenever user input fails to meet the validation requirements.

There are 5 types of validation web controls in ASP.NET with each control performing a specific type of validation [13]

Validation Control	***Purpose***
RequiredFieldValidator	Ensures that data is entered for all required input fields
CompareValidator	• Compares the value of one input with the value of another user input/constant. Hence, offering cross-field validation. • Also used to perform data type validation: ensures that the data type entered corresponds to the requirement i.e. String, Integer, Double, Date, Currency.
RangeValidator	Verifies if the received input is within the valid range of values.
RegularExpressionValidator	• Also referred to as pattern validation. • Used to determine whether the user's input corresponds to a particular pattern.
CustomValidator	Enables customised validation logic for user input.

Table 7: Validation Web controls in ASP.NET

Discussion and Conclusion

In classic ASP, developers had to write separate validation routines and re-use these routines in various parts of an ASP script or other ASP scripts that need to employ form validation. The ASP.NET validation web controls help address form validation needs in a single instance. They are even capable of validating specific sets of input via its Validation Groups option. Therefore, input fields can be grouped together to perform particular validation checks and are reusable across multiple web forms.

ASP.NET also includes Ajax like features in its validation mechanism hence providing both client-side and server-side validation. Generally, validation using the validation web controls takes place on the server regardless if client-side validation is enabled. Client-side validation is enabled by default and can be optionally disabled for the entire web form or for specific controls.

In comparison with the frameworks in Java, the purpose of validation web controls has very straightforward use. It checks if user input from forms is captured in terms of its type, length, format and range. All other incorrect or unacceptable input is rejected with an appropriate error message. Developers can also write customised validation functions to perform more complex validation logic.

However, to validate input from sources like URL/query strings, cookies, HTTP headers, files, and others, developers need to use ASP.NET's *Regex class*. This class is explained in Section 3.2.1.3.

3.2.1.1 Buffer Overflows

Like Java, ASP.NET also comes pre-built with features that help in relieving buffer overflow vulnerabilities. Features such as

1. Type Safety and Code Verification
2. Automatic memory management
3. Garbage Collection System

are offered by the Common Language Runtime (CLR) in ASP.NET.

1. Type Safety and Code Verification

Type safety prevents programs from accessing unauthorized memory locations. Type safety in .NET is preserved by the CLR. During compilation, code written in C#, VB.NET, Jscript.NET, are converted into MSIL (Microsoft Intermediate Language) code. At runtime, the Just-In-Time (JIT) compiler in CLR converts the compiled code (MSIL and metadata) to code native to the operating system. Optionally, a verification process is carried out to examine if the metadata and MSIL are type safe i.e. confirm that the code can access memory locations and call methods that have properly defined types. This process, however, can be skipped if the code has permission to bypass verification.

With the information found in MSIL and metadata, the CLR is able to make sure that references always refer to compatible types, null references are never accessed, and instances are never referenced after they are freed. When code is not type safe, the runtime cannot prevent unsafe code from performing malicious operations. The runtime's security mechanism ensures that it does not access native code unless it has permission to do so. Type safety feature also isolate objects from each other, hence protecting them from being corrupted.

2. Automatic memory management

When a program is loaded into memory, the runtime allots in approximation the amount of address space required by the program. This reserved address space is called the managed heap. The managed heap maintains a pointer to the address where the next object in the heap will be executed. The CLR provides automatic memory management for its managed heap. Automatic memory management eliminates problems related to memory leakage, or attempting to access memory for an object that no longer exists.

3. Garbage Collection System

A process known as garbage collection is used to release memory space which is no longer referenced by objects. As long as address space is available, the garbage collector continues to allocate space for new objects. When the garbage collector's optimizing engine detects no space in the managed heap, the garbage collector thread will be triggered with the highest priority and all unreferenced objects are collected and released from memory.

Discussion and Conclusion

Like the JVM, the CLR provides necessary mechanisms to control buffer overflow vulnerabilities. It manages code at execution time and provides core services like type safety and code verification, memory management, thread management, and garbage collection that promote security and robustness in .NET applications.

To obtain maximum benefits from the CLR, it is essential that a good language compiler is used. The language compiler determines which runtime features are available and establishes the syntax of the application code. In order to reliably depend on the CLR and its associated compiler, developers must make sure the both the CLR and compiler are constantly updated with latest releases which contain more enhanced feature and functionality.

The garbage collection system, implemented as a separate thread in .NET, may be of concern to some developers with regards to the application's performance. Although running a separate thread will create extra overhead, in practice the garbage collector thread is given the lowest priority. Therefore, when the system detects that the managed heap is running out of space, the garbage collector is given REALTIME priority which is the highest priority in Windows systems.

More explicit coding policies to avoid buffer overflows vulnerabilities can be found in Section 4.1.1.

3.2.1.2 Cross Site Scripting

Important countermeasures to prevent XSS attacks, as explained in Section 3.1.1.2 are:

- Filtering and validating all user input
- Setting the character set and language locale for each page generated by the web server
- Encoding all special characters with its equivalent ASCII/UNICODE/ISO format
- Filtering and encoding output of all data types especially for special characters

Input validation in ASP.NET can be done using the validation web controls and Regex class elaborated in Section 3.2.1 and Section 3.2.1.3.

To limit the ways in which malicious users use canonicalization to trick input validation routines, all dynamically generated web pages must be specified with a character set. The character set of an application is defined in the *requestEncoding* and *responseEncoding* attributes of the ***<globalization>*** element in the web.config file.

By default, the *request validation* component in ASP.NET detects any HTML elements and reserved characters that are posted to the server. This helps to prevent users from inserting scripts into an application. *Request validation* is also able to check all input data against a list of potentially dangerous elements and reserved characters. If a match occurs, it throws an exception.

Additionally, developers coding applications in .NET, should use the

1. HttpUtility.HtmlEncode, and
2. HttpUtility.UrlEncode

to impose encoding mechanisms in web applications. With encoded data, all user input or dynamically generated output renders pure harmless web pages.

1. HttpUtility.HtmlEncode

HtmlEncode ensures that input characters including tag attributes that have special meanings in HTML are encoded. As shown in Table 4 of Section 3.1.1.2. This method assures that data is deemed safe prior to being displayed. Alternatively, with the existence of the StringBuilder class in ASP.NET, input containing permitted HTML elements like <b> </b> <i> <\i> and others are not encoded. The StringBuilder class acts like a filter class which allows support for simple text formatting options. A recommended practice, however, is to restrict formatting to safe HTML elements only. Please see Section 4.1.2 for more details.

2. HttpUtility.UrlEncode

This method is used to encode URLs constructed from user input or that contains data received from the client or a shared database. Every parameter and value, more specifically characters that are not allowed in a URL, is properly encoded according to the specified character set.

Discussion and Conclusion

Although HTMLEncode and URLEncode has been around for some time, its main purpose was for handling rendering issues. With the rise of XSS vulnerabilities, these classes have now become more useful.

Nevertheless, additional effort must be taken by developers to prevent XSS. For example, in cases where some HTML tags are permitted, the application must also take care of the attribute and value elements associated with the tag. For example, the <IMG> tag has the *alt* and *src* attribute with its respective text and URL value. Filtering must be carried out to reject and remove unknown tags, attributes and values. However, if the <SCRIPT> tag is allowed, extra caution must be taken and it still may be very hard to prevent XSS vulnerabilities. The documentation on the StringBuilder class to cater for the attribute and value elements of a HTML tag is not very clear.

Similar to Java, developers must take care of the order of encoding and decoding and to ensure that the application does not perform separate encoding and decoding functions in different parts of the application. In the same way, all input data should be decoded before being stored and encoded again before displaying to the user.

Please refer to Section 4.1.2 for more details on avoiding XSS vulnerabilities.

3.2.1.3 Injection Flaws

Input validation is essential to protect an application from malicious command injections. The validation web controls presented in Section 3.2.1 is concentrated on constraining input specifically from web forms. What about validating input received from sources like URL/query string, cookies, HTTP headers, files, and others? Additional measures are required to address this.

The goal of this category is to

- Filter and validate input received from sources other than web forms
- Include validation methods like data conversion and regular expressions

Similar to Java, ASP.NET also contains a regular expression class; *Regex* class. This class is useful for validating input coming from sources like URL/query strings, cookies, and HTTP headers. It provides more thorough input formatting rules to detect uniquely crafted input that otherwise might bypass a standard or incomplete validation routine. Although a version of the Regex class is available for use in validator web controls, i.e. *RegularExpressionValidator*, it is exclusively used for validating input received from web forms.

1. Regex class

The Regex class which resides in the *System.Text.RegularExpressions* namespace is particularly useful for validating input from sources other than web forms such as query strings, cookies, files, and HTTP headers. It specifies a pre-defined pattern to which the input must satisfy. For example, it can be used to ensure that a web address (URL) conforms to the right format and points to an authorized web server serving the application.

However, as mentioned in Java, there are some downsides with the extensive use of regular expressions. Firstly, when sharing large input across applications, regular expressions will consume a lot of processing time which will slow down the response received from the server. One solution for this is to pre-compile the expressions in the applications own assembly (MSIL instructions). This is done by using the static *CompileToAssembly* method on the *Regex* class. The assembly is then added as a reference to which the application will execute when required.

Discussion and Conclusion

In most application, performance is of utmost importance. Although pre-compiling the expressions provide a solution for the application to perform faster, much caution must be taken when using this technique. MSIL instructions are stored in memory after compilation. Hence, if large numbers of expressions are compiled at one time, the amount of heap resources used and not released will be large. The MSIL instructions should not be loaded into the default application domain, because then it cannot be unloaded from memory while the process is running. Developers need to make sure that *AppDomain* objects; used to isolate, unload, and provide security boundaries for executing managed code are unloaded without affecting the process. This is crucial for processes running for long periods without restarting.

Concision is important when it comes to crafting regular expressions. Carefully written regular expressions can prevent against malicious text manipulation. However, most of the time, regular expressions are easier to write than they are to read. Therefore, for large applications, proper documentation on the pattern and code written is essential for maintaining or updating the expressions. This could be one of the reasons why regular expressions are not widely used for preventing injection flaws as its format can get very complex and difficult to understand.

The use of regular expressions to prevent Injection Flaws in Java and .NET are the same. More measure to control injection attacks can be found in Section 4.1.3.

3.2.1.3.1 SQL Injection

SQL injection occurs when malicious users insert unsafe/reserved characters or command strings to construct dynamic SQL statements. These unsafe characters or command strings which are processed at a data source or database server eventually impacts the integrity of the stored data and the application's database. It can retrieve private information as well as modify or destroy information. Just like in Java, to defend against SQL injection vulnerabilities, developers are encouraged to use

- prepared statements with parameterized SQL when querying a database, or
- stored procedures with parameterized SQL when querying a database, and
- a least privileged account that has restricted permissions in the database

The above mentioned measures are implemented in ASP.NET using the *SqlCommand* and *SqlParameter* class.

1. SqlCommand and SqlParameter class

The *SqlCommand* class represents the SQL statements used to query an SQL database. The *SqlParamter* class, on the other hand, presents a parameter to the *SqlCommand* class. Another important class, the *SqlParameterCollection* is used to represent a collection of parameters associated with an SQL query. The *SqlParameter* comes with built-in type checking and length validation function for all parameters. The parameters refer to inputs received from the client and are treated as literal values instead of executable commands. If the inputs received are outside of the type and length range, the *SqlParameter* class throws an exception.

Different from the conventional concatenated SQL strings, prepared statements in ASP.NET will automatically escape characters that have special meaning in SQL before querying a database. Stored procedures are pre-defined SQL statements stored in a database.

Prepared statements and stored procedures on their own cannot prevent SQL injection attacks. They both must use parameterized SQL that will hold the value of the user input. The values are filtered before passing to the parameters. Parameterized queries can be used in 3 steps

- Construct the *SQLCommand* command string using parameter placeholders
- Declare a *SQLParameter* object
- Associate an *SQLParameter* object with an *SQLCommand* object

A parameter uses the @ symbol at the beginning of its parameter name. For example,

```
SqlCommand cmd = new SqlCommand("select * from Country where code =
@Country", conn);
```

Many parameters can be used in a single query. Each defined parameter will match a *SqlParameter* object to a *SqlCommand* object.

```
SqlParameter param  = new SqlParameter();
param.ParameterName = "@Country";
param.Value         = inputCountry;
```

The parameter name in the *SqlParameter* object must be exactly the same as the parameter name used in the *SqlCommand* command string. The value corresponds to the input received from the client. When the *SqlCommand* object executes, the parameter will be replaced with this value.

In prepared statements, these parameters replace the "?" placeholders. For example,

```
        String selectStatement = "select * from Country where code = ? ";
```

becomes

```
     String selectStatement = "select * from Country where code = @Country ";
```

For stored procedures, the *SqlCommand* object needs to know which stored procedure to execute

```
SqlCommand cmd  = new SqlCommand("sp_displayallcountries", conn);
cmd.CommandType = CommandType.StoredProcedure;
cmd.Parameters.Add(new SqlParameter("@Country", country));
```

The sp_displayallcountries is the name of the stored procedure in the database. The second line is the connection object which is used for executing query strings. It tells the *SqlCommand* object what type of command it will execute by setting its *CommandType* property to *StoredProcedure*. The third line adds a parameter to the command which will be passed to the stored procedure.

Stored procedures can further secure a database by restricting objects within the database to specific accounts, for instance permitting the accounts to only execute authorized stored procedures.

Discussion and Conclusion

The solution presented here has the same concept as the solutions in Java. Nonetheless, there are a few things that must be kept in mind to prevent SQL injections in both Java and .NET.

Different database platforms respond differently to certain special characters. For example, in MySQL, the reaction differs for '\b' which is interpreted as a backspace, and '\B' which is interpreted as 'B'. Moreover, not all database support the use of SQL reserved words like UNION SELECT. Hence, developers need to read the database documentation to identify every possible special character and reserved words that has special meaning for that particular database.

Programmers may feel uncomfortable with leaving the security of a program to the settings of a database server when using stored procedures. This is because database settings are often outside the control of the programmer. This may lead to issues of portability and flexibility at the expense of security.

3.2.2 Authentication and Authorization

As iterated in both Chapter 2 and Section 3.1.2, Authentication and Authorization are important to protect against compromising sensitive information to unauthorized users. ASP.NET in conjunction with Microsoft's Internet Information Services (IIS) contributes two levels of Authentication and Authorization controls in .NET based web applications.

When a user requests for a specific resource, the request will be attended by the IIS first. IIS authenticates the user and if successful, IIS hands off the request as well as a security token to the ASP.NET engine for the second level of authentication. Similarly, the ASP.NET engine checks whether the authenticated user is authorized to access these resources. If the authentication succeeds, ASP.NET serves the request; otherwise an "access-denied" error message is sent to the user.

Authentication in ASP.NET applications is accomplished using

1. IIS Authentication, and
2. ASP.NET Authentication providers

Once the authentication process completes, the authorization process takes place. In ASP.NET, there are two ways in which authorization takes place:

3. File authorization
4. URL authorization

All Authentication and Authorization configuration settings can be found in the IIS metabase and Web.config file.

1. IIS Authentication Methods

IIS provides a few different ways for authenticating a user identity;

Basic Authentication	• Transmits credentials (e.g. username and password) across the network in an unencrypted form. • Uses the web server's encryption features to secure information transmitted across the network.
Digest Authentication	• Transmits credentials across the network as an MD5 hash, or message digest, where the original username and password cannot be deciphered from the hash.
Integrated Windows Authentication (NTLM or Kerberos)	• Credentials are hashed/encrypted before being sent across the network. • Knowledge of the credentials is proven through a cryptographic exchange with the web server.
Client Certificate based Authentication	• Creates and uses digital certificates to authenticate users without having to provide credentials each time they log on.
Anonymous Authentication	• No authentication takes place

Table 8: IIS Authentication Methods

A particular method or even the combination of one or more methods is chosen per use in the IIS administrative services.

2. Authentication Providers

Authentication providers perform authentication on the basis of principals and credentials. A client's (user, system, process) identity is referred to as a security principal. Credentials are used to verify the identity of the principal. Upon successful authentication and authorization, the principal is able to access the resources on the system. ASP.NET also supports custom authentication providers.

There are three (3) types of authentication providers in ASP.NET.

a) Form Authentication

This is a cookie based authentication implementation where credentials are stored in a text file or a database. Using this form of authentication, developers can specify which files on the site can be accessed and by whom, and allows identification via a login page. If login is successful, ASP.NET issues a cookie to the user and automatically redirects the user to the requested resource. This cookie holds authentication information that allows the user to revisit authorized resources without having to repeatedly log in for the lifetime of the session.

b) Passport Authentication

This is a centralised authentication service developed exclusively by Microsoft which offers a single sign-on facility for its web applications. Previously, Passport authentication provided limited support for use on other platforms. However, Microsoft has withdrawn partner Passport usage making this authentication mechanism no longer a viable option for web applications communicating across multiple platforms.

For general knowledge, this service requires application servers to be connected with Microsoft's Passport servers. When using these options users need only to remember one username and password pair to login and access all partner sites. Examples of applications with Passport enabled account are MSN.com accounts.

c) Windows Authentication

This is the default authentication mechanism for ASP.NET applications and is used in parallel with the authentication mechanism provided by IIS. Applications' adopting this method of authentication eases the coding effort required as it requires minimum ASP.NET coding. The authentication relies on Internet Information Services (IIS) to authenticate the user.

The impersonation element configured in the IIS for Windows authentication enables ASP.NET applications to optionally execute with the identity of the client, that have already been authenticated by IIS. The reason for this is to avoid dealing with authentication and authorization issues in the ASP.NET application code. Upon successful authentication, IIS passes an authenticated token to the ASP.NET application. Otherwise, IIS passes an unauthenticated token which means that it wasn't able to authenticate the user.

The ASP.NET application relies on the settings in the NTFS directories and files to allow it to gain or deny access.

A graphical representation of the authentication options between IIS and ASP.NET is as shown

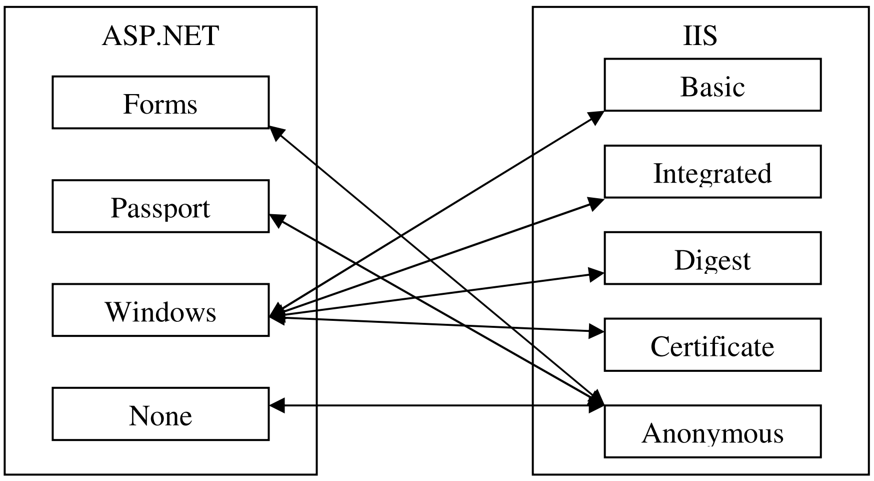


Figure 17: Authentication options between IIS and ASP.NET

Next, we take a look at the Authorization checks offered in ASP.NET.

3. ***File Authorization***
   File authorization uses the Access Control List (ACL) of the resources (.aspx or .asmx extensions) to determine whether an authenticated user is authorize to access the resources. Windows ACL allows file permissions to be set on application files. However, this solution only works if the Windows authentication with impersonation is used.

4. ***URL Authorization***
   This module associates users and roles to URLs. It selectively permits or denies access to arbitrary parts (directories/subdirectories) of an application to specific users/roles. For example, ASP.NET checks whether the user has access to /Default.aspx (). Authorization rules specifying the access rights for user/groups are configured in the Web.config <authorization> element.

Discussion and Conclusion

Having seen the types of authorization and authentication checks available in ASP.NET, there are still a few things that developers cannot afford to forget when using these authorization and authentication schemes. Some reminders include

- Form Authentication uses custom HTML forms to collect authentication information. Developers need to write additional logic/code to check the credentials against a database or some other data store.
- With Windows authentication and File Authorization, the developer must be sure to format the server file space as NTFS so that access permissions can be set.

A few downsides in the authentication mechanisms are

- Form Authentication only restricts access to ASP.NET files and not to static or ASP (classic) files unless those resources are mapped to ASP.NET file name extensions.
- When IIS receives a request for an ASP.NET application, the IIS settings are applied first regardless of the ASP.NET configuration settings for that request. This can result in a request being inaccessible to users or have less restrictive security settings.
- The use of certificate based logon leads to issues like management of certificates on browsers which are non-trivial. Moreover, it is expensive to maintain a large database of certificates with a large number of users.

Analyzing the authorization mechanisms, we find that

- The File Authorization module only performs checks for user requested files and not for files accessed by the code. Files accessed by code are dependent on IIS for access control. This form of authorization requires developers to set access control permissions (i.e. ACL) on all resources associated with the application, which can be a cumbersome effort.
- Even though the ASP.NET resources in a directory might be restricted by the Web.config file, all users can still view the files located in that directory if directory browsing is turned on and no other restrictions are in place. This may not be a very good mechanism as attackers might have the knowledge of what files are available in which directories. Further measures must be put included.

Similar to Java, ASP.NET doesn't relate authentication and authorization to a particular session. Associating sessions to an authentication and authorization scheme is beneficial in order to proceed with subsequent requests from the same client.

Between the JAAS package and the authentication and authorization solutions in ASP.NET, although the JAAS package still requires additional programming effort, it seems to be a more promising authentication and authorization mechanism for web applications. The ASP.NET solution seems to be only compatible for applications using Windows platform and is probably not able to cater for larger cross-platform integration.

3.2.3 Improper Error Handling and Logging

All applications at some point will contain errors or undergo unexpected situations. It is therefore very important that an application is able to identify where errors might likely occur and write code to anticipate and handle them. A good application is able to capture errors early in its execution.

Error and exceptions in ASP.NET can be divided into two separate logics:

1. Redirecting the user to an error page when unforeseen errors occur. There are two different pages to which users are redirected to:
   - Page level (applies to errors that happen within a single page)
   - Application level (applies to errors that happen anywhere in the application)
2. Handling the exceptions as programmed in the application.

1. Error Handling

If an error is due to a fault by the user (i.e. wrong input) the application should redirect the user to the same page prior to the error, with an appropriate error message informing the user of the next step. ASP.NET provides three methods, executed in the following order to trap and respond to errors when they occur:

a. Page_Error event handler in the aspx file
b. Application_Error sub in the global.asax file
c. customErrors section of the web.config file

The *Page_Error* event handler traps errors that occur at the page level. The application is programmed to display error information or log the event or perform any other desired action.

The global.asax file handles custom errors at the application level. Errors can also be logged and redirected to another page. It is basically the same as the Page_Error handler but happens to be at the application level rather than the page level.

The *customErrors* section is used to restrict display of detailed error messages. It holds three different attributes

- defaultRedirect - specifies the URL to redirect a browser, if any unexpected error occurs.
- subtag <error> - specifies the error status code before redirecting to a specific page.
- statusCode - specifies the error status code and the redirect attribute that states the URL of the redirect page

2. *Exception Handling*

Exception handling, which uses the Try, Catch and Finally construct is useful in handling abnormal situations. The try statement generates the exception, the catch statement handles the exception from a central location and the finally statement closes or removes resources associated with the exception.

When using this construct, it is important to remember the order of functions in the catch code which has different possible outcomes. Exceptions in catch blocks must be ordered from the most specific to the least specific. Hence, specific exceptions are given more priority before the more general catch block.

The Exception class in ASP.NET is a member of the System namespace and is the base class for all exception occurrences. It consists of two main subclasses; SystemException class (base class for all run-time generated errors) and the ApplicationException class (used when non-fatal application error takes place). An example of classes in the SystemException class includes the IndexOutOfRangeException class (this defends off-by-one error vulnerabilities), Null Reference Exception class, InvalidOperationException class. With ApplicationException classes, developers can create customised exceptions.

3. *Logging*

With respect to logging, the ASP.NET Health Monitoring feature enables system administrators to monitor the status of deployed web applications. It not only log events that relate to errors but all other events related to performance, security, tracing, debugging that is useful to examine.

Discussion and Conclusion

ASP.NET is seen to have a comparable mechanism with Java to handle and log erroneous and exceptional events that takes place within an application. Moreover, the language independence of the .NET framework allows error handling in one language (i.e. VB.NET) to be handled also in another language (i.e. C#).

Nonetheless, when displaying error messages, care must be taken to not give away information that might be helpful for malicious users. Developers still need to envisage all possible errors that could occur in an application and for all other events the application is able to react accordingly.

For logging, it is important to ensure the format of information log. Furthermore, the type of information logged must also be clearly specified by the developers. The application should not log user sensitive information. Developers should make sure that the log files are kept in restricted locations and all sensitive information within it are encrypted. More coding policies and guidelines for logging web applications can be found in Section 4.4.

3.2.4 Insecure Storage

Part of web application security is to ensure that highly sensitive information like passwords, connection strings, encryption keys and the like are not retained in a readable or easily decoded format. Hence, a trustworthy storage mechanism is required to disable access to protected information.

.NET provides various means to ensure the security of stored information including

1. Cryptography provided by the System.Security.Cryptography namespace
2. Configuration settings in the Web.config file
3. Data Protection API (DPAPI)

1. System.Security.Cryptography

Encryption uses cryptography to protect data from being viewed or modified by unauthorized users. ASP.NET's cryptographic solution is provided via the System.Security.Cryptography namespace. This namespace contains classes that can perform symmetric and asymmetric cryptography, create hashes, digital signatures, signed and/or enveloped messages and random number generation.

2. Configuration settings in the Web.config file

Apart from files and databases, ASP.NET also stores sensitive information in configuration files. To secure information in configuration files, ASP.NET provides a feature called protected configuration in the Web.config file, which enables the encryption of sensitive information in a configuration file.

Information that is especially sensitive includes the encryption keys that are stored in the machineKey configuration element and the connection strings stored in the connectionStrings configuration element which provides access to a data source.

3. Data Protection API (DPAPI)

This is a cryptographic API, offered only in the Windows operation system (since Windows 2002) that is slowly gaining popularity. It allows the encryption of data using information from the current user account or computer. It uses the underlying Windows password infrastructure to avoid explicit key storage.

DPAPI implementation alleviates the difficult problem of explicitly generating and storing cryptographic keys. Moreover with DPAPI, application developers need not write specific cryptographic code to protect sensitive application data like passwords and keys.
The DPAPI is analogous to the Java KeyStore used in Java applications.

Discussion and Conclusion

All encryption techniques require the use of keys. More often, the issues is how secure is the storage mechanism for these encryption keys. If the key is to be encrypted, another key is needed, and it goes on. The existence of DPAPI in ASP.NET allows developers to encrypt keys based on a particular user's profile or all the users of the local machine using the system DPAPI key. This is an interesting method which may be difficult to compromise as the key is stored in the operating system's registry. Furthermore the location in the registry can also be set by the developer further strengthening the security of the key.

In all cases, when using encryption techniques it is important to understand the risks associated with key management. Application designers and developers must be made aware of the

- level of protection required on the keys,
- method to distribute and manage keys among users,
- issues involved in activating, changing or updating keys
- archiving keys
- revoking, deactivating, and destroying keys

It is also imperative to outline recovery procedures for encrypted information in the case of lost, compromised or damaged keys. Moreover, to reduce the likelihood of compromise, activation and deactivation dates for keys should be clearly defined so that the keys can only be used for a limited period of time.

3.2.5 Application Denial of Service

Application Denial of Service (DoS) is used by malicious users to compromise a web application by making its services unavailable or inaccessible. This is done by utilizing large amounts of application resources like memory, CPU, bandwidth and disk space.

Like Java, there are no specific APIs/frameworks in .NET to prevent application DoS attacks. Some suggested countermeasures include

- Mandating the use of try-catch-finally blocks for handling errors and exceptions
- Configure IIS to prevent an application from using disproportionate amount of CPU time, memory, bandwidth, disk space and any other resources
- Perform thorough input validation using the techniques discussed
- Limit the number of queries sent to the database
- Incorporate all SQL Injection prevention measures
- Limit the number and size of file uploads and form posts. This can be done by setting the maxRequestLength value (in kilobytes) and/or RequestLengthDiskThreshold in the Web.config file

Discussion and Conclusion

Application DoS occurs not only because of flaws existing in code but also unethical intent of malicious users which coding policies may not be able to resolve. The above mentioned prevention methods does not provide a perfect solution as it is quite impossible to counter this form of attack. By constantly studying the operation or functions of an application, an attacker can still find means to perform an application DoS.

For example an attacker can take advantage of a login page by submitting multiple usernames and eventually lock out multiple legitimate users. Even if the application implements a delay after a fixed number of login failures, the application is still inaccessible to the legitimate users. Similarly, applications that have complex regular expressions rules are also susceptible to application DoS. An attacker can send in many different types of text patterns which will consume longer processing time by the application.

The suggested countermeasures should be applied in all web applications whether developed in Java, .NET, PHP, etc. The difference is only in the implementation method.

3.2.6 Configuration Management

As elaborated in previous chapters, configuration management is concerned with managing the contents of a web site, its storage media and directory services, the tools and procedures for accessing configuration information, its connection with each other and to external systems or other servers. Improper or inadequate configuration management activities can lead to many security problems.

Configuration management must be exercised at all stages of an application development lifecycle especially during the development, deployment and maintenance stage. Properly documented code, directories, data storage can particularly ease the task of developers and site administrators besides protecting the applications from being misused.

ASP.NET makes available configuration management functionalities for all its servers and application. Configuration features include access control, encrypted connection strings, page caching, compiler options, debug and trace options and many others. All of these and more are provided via two main configuration files but can also be administered by its graphical user interface tool, the Microsoft Management Console (MMC) snap-in.

All configuration information in ASP.NET is stored in the Web.config and Machine.config files. The Machine.config file is used for configuring settings of the server. An ASP.NET application initially inherits the default configuration settings from the Machine.config file. The Web.config file on the other hand includes application specific configuration information. The Web.config file can appear in multiple directories in an ASP.NET application while the Machine.config file is stored in the configuration directory of the install root. If the web application spans multiple folders, each sub folder has its own Web.config file that inherits or overrides the parent's file settings.

The following lists some of ASP.NET configuration management features

- Protect configuration files from unauthorized access by configuring code access security. An administrator can explicitly state which protected resources an application can access, which version of assemblies an application will use and where remote applications and objects are located.
- Deny access attempts to any browser requesting for the Machine.config or Web.config files.
- The configuration settings of each ASP.NET application are independent of each other. Configuration files on one application cannot access the configuration settings of another ASP.NET application. However, if applications are configured to run in full trust, then the application has permission to read the configuration files of other applications.
- Enables parts of the Web.config file to be locked. This prevents configuration information from being overwritten.
- Encryption option for sensitive/protected data stored in the Web.config file.
- Disables remote administration options by default. If enabled, only authenticated users are authorized to read or write configuration data.

Discussion and Conclusion

Configuration management must be practiced from the moment any work on an application begins to when the application ceases to operate. The .NET framework provides a large amount of control and flexibility options to its developers and site administrators. Nonetheless, the features of the ASP.NET configuration system only apply to ASP.NET resources.

Also the configuration of an ASP.NET application is dependent on the IIS configuration. Hence, it is compulsory to ensure proper patching and updates between the application and web server, and any other connection to external systems. Developers need to make sure that all updates should not leave any open gaps.

Analyzing the configuration facilities and the current vulnerabilities associated with configuration management, ASP.NET applications still require additional process to control configuration management activities. This includes processes like file naming conventions, tracking and controlling changes in documents, document baseline procedures, versioning history, backup/archiving of files, recovery from configuration management errors, restoration from disaster recovery, build management, release control and many others. As mentioned in Section 3.1.6, organizations can use one of the many configuration management tools that assist in handling configuration issues.

Summary on ASP.NET

The .NET framework like Java, provides a wide range of security measures that are able to defend against vulnerabilities described in Chapter 2. Briefly, for input validation, .NET applications can use the *validation web controls* for validating user input from web forms; cross site scripting vulnerabilities can be handled using the *request validation* component, as well as the *HttpUtility.HtmlEncode*, *HttpUtility.UrlEncode* methods; injection flaws can be avoided by using the *Regex* class; similar to Java, SQL injection is prevented with the use of the *SqlCommand* and *SqlParameter* class; authentication and authorization issues are managed by both the *IIS* (Basic Authentication, Digest Authentication, Integrated Windows Authentication, Client Certificate Authentication, Anonymous Authentication), A*SP.NET Authentication Providers* (Form Authentication, Windows Authentication) and *File Authorization* and *URL Authorization* ; errors are handled using the *Page_Error*, *Application_Error* and *customErrors* event handlers while exceptions are managed with the *SystemException* and *ApplicationException* class; logging in ASP.NET can be achieved via *ASP.NET's Health Monitoring feature*; insecure storage can be prevented by using cryptography (*System.Security.Cryptography*), by setting the configuration in the *Web.config file* and incorporating the *Data Protection API* (DPAPI); and configuration management issues can be taken care by *Microsoft's Management Console (MMC) snap-in* as well as the settings specified in the *Web.config* and *Machine.config* files. Just like Java, .NET includes built-in mechanisms through its Common Language Runtime (CLR) to prevent the occurrences of buffer overflow vulnerabilities. There are no specific API/frameworks in .NET to prevent application DoS attacks. Most application DoS attacks can be controlled by implementing the preventive measures of all other vulnerabilities.

Although one of the principal aims of .NET is to be used across multiple platforms, just like Java, much of its security controls seem to be only applicable for applications in the same environment. This is clearly exhibited in its authentication and authorization mechanisms where most options are very dependent on the Windows operating system. Perhaps efforts for total platform independence are still underway and possibly made available more extensively in the not too late future.

ASP.NET seems to provide more easily available APIs, tools and framework, which come with adequate documentation on its purpose and usage. Instructions given in ASP.NET are clear, consistent and easy to follow which makes it a preferred choice of language for entry-level programmers. Nonetheless, developers still need to be very careful when using these APIs, tools and framework so that they don't miss any important steps that might have been excluded from the documentation.

Now that we've seen the threats and vulnerabilities as well as the available prevention mechanisms in the two most widely used web development language, we now move on to the final chapter which contributes to the major part of this thesis.

4 Coding Policies and Guidelines

Having identified the types of threats and vulnerabilities as well as attack scenarios in web applications (Chapter 2) and the existing libraries/APIs/frameworks in both the Java EE and .NET framework (Chapter 3); this chapter intends to present a set of secure coding policies and guidelines for web developers implementing web applications. These policies and guidelines can be applied in any of the web programming languages (Java, ASP.NET, PHP, Perl). The coding policies and guidelines result from the extensive literature study and analysis performed on existing web application threats and vulnerabilities. Other relevant security threats and vulnerabilities which is the outcome of a much thought-out assessment have also been included. The main objective of this chapter is to provide a more structured approach together with a comprehensive list of coding policies and guidelines that would result in more robust and secure web applications.

The coding policies and guidelines are organized in the following manner:

Synopsis	Gives a brief summary about the threat/vulnerability
Controls	Defines measures that must be taken to avoid attacks caused by such threats/vulnerabilities
Guidelines	States explicit implementation policies and guidelines to develop secure web applications
Supplementary Information	Outlines any additional information which may be useful for web developers

The major references used in building this policies and guidelines are as follows:

1. OWASP Foundation. *A Guide to Building Secure Web Applications and Web Services.*2.0 Black Hat Edition. July 27, 2005.
2. ISO/IEC 17799. *Information technology – Security techniques – Code of practice for information security management.* Second Edition. August 15, 2005.
3. Mcclure S., Scambray J., Kurtz G. *Hacking Exposed: Network Security Secrets and Solutions.* Fifth Edition. McGraw-Hill 2005. Chapters 11 – 13.
4. Andrews M., Whittaker J.A. *How to Break Web Software.* Addison-Wesley 2006.

4.1 Input Validation

Synopsis

Proper input validation is the strongest measure of defense against today's web application attacks. The study in the preceding chapters demonstrated that the majority of application level attacks come from maliciously formed input. Inputs from web applications are received from a variety of sources ranging from web forms, cookies, headers, URL/query string parameters, databases, and other data sources, which may be trusted or untrusted sources. All of these input data play a role in the applications' processing. Therefore, to mitigate malicious input from compromising an application the following controls must be adopted. Suppose an application receives an unexpected input that doesn't conform to any of the controls, the best course of action is to raise and error, log the event and stop processing immediately.

Controls

- Use both client side and server side validation techniques or alternatively techniques like Ajax
- Ensure that all inputs are correct in terms of type, length, format and range
- Prevent users from entering incorrect or unacceptable values

Implementation Guidelines

(*M: mandatory*, *O: Optional*)

Tag	Policy	M/O	Related Tag(s)
General Policies			
4.1	Be sure that the application architecture mandates the use of SSL/TLS technology. The setting is done in the web server running the application. *This forces all communication channels between the web client and web server to be encrypted; preventing eavesdropping, tampering and message forgery vulnerabilities.*	M	
4.2	Unless carefully designed and programmed, web pages that use the SSL/TLS technology should ***NOT*** automatically switch between the HTTP and HTTPS protocol. *For example,* ***Apache*** – Define HTTPS connections in Apache mod_ssl module HTTPConnection con = new HTTPConnection("https" , www.webAddress.net, -1); ***ASP.NET*** – Define HTTPS connections in the ASP.NET Web.config file, between the <secureWebPages> and <\secureWebPages> tag.	M	
Numeric/Alpha/Alphanumeric Fields			
4.3	Determine if inputs are required or optional. – Required inputs are inputs that the user must provide. – Optional inputs are inputs that the user may choose to either provide or not to provide. *This is necessary to ensure that users don't leave required fields empty.*	M	
4.4	Always define a minimum and maximum length for all input field types. *e.g. credit card: minimum length 14, maximum length 16*	M	
4.5	Check if the input is in its allowed data type. *i.e. numeric, alpha, and alphanumeric*	M	
4.6	Check if the input is in its allowed format/syntax. Some examples: *1. For numeric input* *– Check its range* *– Determine if it's signed/unsigned* *2. Email validation* *– The email includes the @ symbol and has correct/acceptable domain names* *– Avoid storing spoofed email addresses by requesting users to click on a confirmation link sent to the given email address*	M	

	3. Credit/Debit card validation *– Determine the accepted credit/debit cards i.e. MasterCard, Visa, Diners, American Express (Amex), Discover* *– Usual form of credit/debit card numbers:* *XXXX XXYY YYYY YYYC* *With C being the checksum, X being the issuing institution and Y the user's card number* *– Four points to consider* ***Prefix matching*** *A list of valid prefixes associated with a credit/debit card. For example, Visa cards must start with the digit "4", MasterCards must start with digits "51,52,53,54, or 55"* ***Length*** *Number of valid digits associated with a card* • *MasterCard: 5500 0000 0000 0004 (16 digits)* • *American Express: 3400 0000 0000 009 (15 digits)* • *Diner's Club: 3000 0000 0000 04 (14 digits)* ***Check digit*** *Validates the authenticity of a credit card number. A simple algorithm (Mod 10 algorithm) is applied. It performs numerical data validation routines on the number provided.* ***Expiration Date*** *Date provided is acceptable and in correct format* *4. Telephone or mobile number validation* *– Distinguish between international and local dialling* *– Accept digits [0-9], minus, parenthesis* *– All numbers should at least be in the (nnn) nnn-nnnn format. Any other formats should be clearly defined*		
4.7	Perform cross field validation for certain inputs. *e.g.* *1. A postcode field in Europe (country field) should contain 4 numeric and 2 alpha characters* *2. A check payment mechanism should include an appropriate bank routing number and bank account* *3. Credit card payment should include a credit card number and an acceptable expiration date*	O	
4.8	Trim white spaces in the beginning and end of each input fields. If white spaces are allowed/required (i.e. in text areas) encode white spaces to its HTML equivalent (%20).	M	
4.9	Runs of white space must be replaced by a single space and encoded to its HTML equivalent (%20).	M	
4.10	Do not allow the use of NULL characters (ASCII 0, UNICODE U+0000) which are typically used to signify the end of strings. *e.g. In Java, the StringUtils class can be used to check if an input contains a null. If a NULL character is detected an exception must be thrown* *If a NULL character (0 or %00) is accepted as an input, it can cause strings to be terminated early.*	M	4.35
4.11	Avoid the use of hidden fields by storing, retrieving and processing hidden field data at the server side. If unavoidable:	M	4.86, 4.111

	– Evaluate if the data it contains is subject to security risks. – Encrypt/hash the information stored in hidden fields.		

4.12	Do not use HTTP Headers to make any security decisions.
	– HTTP Header Referer normally contains the URL from where the request originated from
	
	`POST /thepage.jsp?var1=page1.html HTTP/1.1`
	`Accept: */*`
	`Referer: http://www.example.com/index.html`
	`Accept-Language: en-us`
	`……`
	
	The contents of the HTTP Header can be manipulated by attackers.
		M	4.75

4.13	Do not allow application to auto-correct wrongly entered input.	M	
4.14	Avoid/minimize use of JavaScript. – If used, o make sure that references to DOM objects are always inspected. *e.g. document.URL, document.location,document.open, and others*	M	
Restrictive Controls (Checkboxes, Radio button, Drop down lists)			
4.15	For all restrictive controls including hidden fields and other elements not directly modifiable by the user, name them using an index/label. – Attach an index/label to the *name/value* attribute of the restrictive controls. The name/value pair with the corresponding index/label is then validated at the server side and incorrect/missing pairs should generate an error message to the user. *For example* – Checkboxes *<input type="checkbox" name="opt1" value="A1"/>* *<input type="checkbox" checked="yes" name="opt2" value="A2" />* – Radio Buttons *<input type="radio" name="sex" value="X1">* *<input type="radio" name="accept" value="J2" checked>* – Drop down Lists *<select name="prdtype">* *<option value="M1">MasterCard</option>* *<option value="M2">Visa</option>* *<option value="M3">Diners</option>* *</select>*	M	4.46
Password Fields			
4.16	Define a minimum and maximum length for the password. – It should consist minimally between 6 to 8 characters.	M	4.4
4.17	Passwords should conform to the following format/syntax: – At least one upper case letter (A-Z) – At least one lower case letter (a-z) – At least one number (0-9) – May include carefully selected special characters like $ % ^ * ( ) _ . / ; [ ] " { } | - – Disallow use of dictionary words	M	4.6
4.18	Set an expiry date for passwords – Typically every 30 to 90 days, depending on the application and its data. – Users should not be able to use the same password (or last 5	M	4.20, 4.22

	passwords) when passwords expire. ○ Retain old hashed/encrypted passwords to prevent password re-use. *The attacker can only access a compromised account until it expires. Changing passwords is often met with resistance because it becomes more difficult for the user to remember. However, complex passwords offer optimal protection.*		
4.19	Implement account lockout policy. – Disable users' account and kill session, if an incorrect password is entered a specified number of times (usually 3-5 times) over a specified period (e.g. last 15/30 minutes). – Implement a delay before allowing a user to re-access locked accounts. *Helps to prevent from password guessing hence decreasing the likelihood of successful attacks.*	M	4.20, 4.77, 4.90
4.20	Forgotten/Change/Unlock passwords Depending on the application, – Encourage the use of secret question(s) and answer(s) (or pass phrases) to retrieve forgotten passwords or confirm user identity. ○ Avoid using fixed/general questions like ▪ What's your pet name? ▪ What's your favourite colour? ○ Allow users to create their own secret question and answers – Use CAPTCHA (Completely Automated Turing Test To Tell Computers and Humans Apart) to prevent automated programs from gaining unauthorized access to accounts. – Provide users the option to unlock accounts by answering their secret question(s)/pass phrases. This is only applicable for applications that do not deal with critical data like banking, medical, government information. – For non critical applications, newly generated passwords may be sent to the users primary/secondary email address. ○ The password is sent along with a timestamp. The user must use the password before it expires which is usually within minutes or a few hours. – Prevent passwords from being changed too frequently. Applications that contain critical data i.e. banks; hospitals; government agencies; may either require the user to reset the password at their nearest branch/location or request for a new/reset password using the conventional systems (phone/mail). ***Note:*** *Since all passwords are stored in a hash/encrypted form, retrieving forgotten passwords is not an easy task. The application should not send hashed/encrypted passwords to the users. Hence, the application sends a newly generated password to the user via email and on the first logon attempt; the user is prompted to change the password. The new password is then hashed/encrypted and stored.*	M	
4.21	Passwords are sent in clear text to the web server. – Security depends on the SSL/TLS technology used.	M	4.1
4.22	Perform one-way hash or encryption before storing the password. – It is important that a strong encryption/hashing algorithm is chosen, together with an appropriate key handling/storage mechanism that is deemed secure. ○ Recommended hashing algorithms include SHA-256, AES-128	M	4.111, 4.112

	in digest mode. ○ Further harden hashed passwords by adding salt (a cryptographically secure random value) to the hash. ○ Recommended encryption algorithms include 3DES, RSA. – When using encryption, keys must be strongly protected to ensure that they cannot be grabbed and used to decrypt the password file. – Salting techniques are useful to make sure that hashed passwords are different even if they coincidently represent the same passwords. – Retain old hashed/encrypted passwords to prevent password re-use. In subsequent accesses by the user, the provided password is compared to the hashed/encrypted password stored in the database. If there is a mismatch, access is denied. *It would require an enormous amount of computing power to find a string which hashes to a chosen value. There's no way to decrypt a secure hash. The uses of secure hashes include digital signatures and challenge-response authentication.*		
Form Submission			
4.23	Use the POST instead of GET action method. Reasons: – POST method sends form input in a data stream and not part of the URL like GET. – Data is not visible in the browser address and hence not recorded in the web server log files. – Although POST information can still be sniffed as it is transmitted across the Internet, sniffing must be done in real time and the attacker needs to have physical access to the data lines between the web browser and web server.	M	
4.24	Data transmitted using the POST method relies on SSL/TLS technology for secure transmission to the web server.	M	4.1
4.25	Prevent forms/transactions from being submitted multiple times from the same user. – Generate a unique, random string and link it with the form or transaction. – Session timeouts/Refresh actions should automatically invalidate the purchase. *For e.g. in e-commerce sites like Amazon.com, EBay and the like, each successful purchase is assigned an ID which is unique per transaction. This ID is stored, logged, and e-mailed to the customer.*	O	

Supplementary Information

1. Typical input fields in a web application are

Alphanumeric data	
Text Field (one line field)	username, address, postcode, email, search keywords
Text Area (multi line field)	message boards, comments/reviews, email messages
Password fields	password
Alpha only data	
Text Field	name
Numeric only data	
Text Field	telephone, credit card, bank account number, ISBN number

List box/Combo box	users select a choice from a list of options e.g. state, country, date (day/month/year)
Radio Buttons	selection of only one item from a mutually exclusive group
Check Boxes	allows multiple selections of listed items

2. Points to remember:
   - For input values used in different methods in the code, it is recommended to assign the input to variables inside the method itself so each thread will have its own copy. If the variables are declared outside of the methods, all threads will share the same copy and may cause some inconsistencies.
3. There are two underlying methods for input validation
   - White Listing: Lists all acceptable inputs
   - Black Listing: Lists all unacceptable inputs

   There is some obvious insecurity in using Black Listing alone. It is difficult to ensure the completeness of unacceptable inputs because black lists are under constant change as new attacking methods are discovered. White lists are more encouraged as it is built by categorizing inputs into various groups (i.e. letters, numbers, alphanumeric characters, punctuations, HTML entities). These groups are then validated against the classes and patterns they form.

4.1.1 Buffer Overflows

Synopsis

Once perceived to be the most notorious vulnerability in applications. However, the properties of high-level programming languages like .NET and Java have become more resilient to this kind of attacks. These languages dynamically check memory and array accesses and automatically resize buffers or free memory space when needed. However, it's best if developers also take careful precautions in order to not solely depend on the languages' properties.

Controls

- Use compiled/interpreted & strongly typed programming languages
- Specifically reserve sufficient amount of space for user input/data on the stack/heap/arrays
- Prevent unauthorized access or tampering on an application's memory space
- Only filtered and validated data are sent to memory

Implementation Guidelines

(*M: mandatory*, *O: Optional*)

Tag	Policy	M/O	Related Tag(s)
4.26	Check length of data before accepting into memory. – Data must be of expected data type/format. – Data must be within the boundaries of defined buffer lengths. *This is to avoid overwriting execution stack and stack pointers.*	M	4.4, 4.5, 4.6, 4.31
4.27	Inspect the properties of the buffers used in the application: – Types of buffer (stacks, heaps, arrays) – Allocate an adequate amount of space initially committed to buffers – Reserve sufficient amount of virtual address space for the buffers – Determine the stack address space reserved, allocated and managed by a user – Determine the size of memory space allocated for a user's stack	M	

4.28	Check the code for use of unsafe functions/keywords. – Certain function calls can have insecure ramifications if used incorrectly. Hence, some functions need careful examination even if the calls are from safe libraries. For example, check if o input pointers is a NULL, o input strings are missing a terminating null character, o the length arguments are correctly defined, o any off-by-one errors, and o any truncation errors.	M	
4.29	Ensure that any URL accessed or resulted from the application is validated before further processing. – Develop customized functions or use regular expressions to specify explicitly the form, length, symbols, and characters, separated by /'s that are appended to a URL and is acceptable to the application. *For example, the following regular expression will only allow URLs starting with www,* *^w{3}\.[0-9a-z\.\?&-_=\+V]+$* *Prevents improperly or maliciously crafted URL.*	M	4.42, 4.118
4.30	Use customized functions or regular expressions to restrict malicious commands/symbols/extensions.	M	4.35, 4.42, 4.43
4.31	Prevent off-by-one errors by – Performing bound checking on arrays – Reviewing conditional statements that o uses mathematical operators like >, <, ++, -- o uses logic operations like AND/OR operators	M	
4.32	Free allocated/reserved memory space after use. – Remove/delete all associated contents/resources.	M	

Supplementary Information

1. Use compiler tools (like StackShield, StackGuard, LibSafe) to detect stack overruns at runtime
2. Make use of the numerous tools available (CodeSurfer, PREfast) to scan source code to find unsafe functions or keywords used.
3. Keep servers, JVM (for Java applications) and CLR (for .NET applications) fully patched with latest updates at all times

4.1.2 Cross Site Scripting

Synopsis

Cross site scripting is a type of input validation vulnerability rooting mainly from representations problems caused by special characters (also known as meta-characters). These characters have special meaning under the HTML specification. As elaborated in previous chapters, this vulnerability may be found on search engines that echo keywords entered in a search, error messages that echo the string containing the error, forms/message boards where users are presented with messages entered by other users.

Controls

- Set the character set/language locale for each page generated by the web server
- Filter and validate all user input

- Encode all input data containing special characters
- Encode all output data with its equivalent ASCII/UNICODE/ISO format

Implementation Guidelines

(*M*: *mandatory*, *O*: *Optional*)

Tag	Policy	M/O	Related Tag(s)

4.33	Specify the character set for the application in the HTTP header:
	
	`Content-Type: text/html; charset=ISO-8859-1`
	
	Examples of popular character sets include ISO 8859-1, Unicode, and ASCII.
	
	The specification of character set should be done at the server end. The setting is independent among the servers.
	Briefly,
	– ***Apache Server:*** This can be done via the *AddCharset*(Apache 1.3.10 or later) AddType directives for directories or individual resources, or *AddDefaultCharset*(Apache 1.3.12 or later).
	– ***IIS 5 and 6:*** *Under the Internet Services Manager → Properties → HTTP Headers → File Types... → New Type...*
	
	The appropriate header can also be set using server side scripting. For example,
	– ***Java Servlets:*** Use the *setContentType* method on the *ServletResponse*
	`resource.setContentType ("text/html;charset=utf-8");`
	– ***JSP:*** Use the page directives
	`<%@ page contentType="text/html; charset=UTF-8" %>`
	– ***ASP.NET:*** `<%Response.charset="utf-8"%>`
		M	4.41

4.34	Set the language locale and country code for each generated web page.	M	

4.35	Encode all special characters into their HTML equivalents using techniques such as *URLEncode* and *HTMLEncode.*
	
	Points to remember: Do not echo/display/store unfiltered input.
	e.g.
	Special characters | ***HTML character entities*** | ***HTML numeric entities***
	< | *&lt;* | *&#60;*
	& | *&amp;* | *&#38;*
	À | *&Agrave;* | *&#192;*
	Σ | *&Sigma;* | *&#931;*
		M	

4.36	Define a list of acceptable HTML tags that may be used in parts of the application. This can be done by – Using a customized filter to encode all accepted tags – Create a separate markup language for the application ○ *e.g. to bold use @B in front of the affected text* – Accept permitted tags automatically and prohibit all others *Example of acceptable tags include:* *<b> <\b> <u> <\u> <i> <\i> <img> etc*	M	4.37

	Some of the potentially dangerous scripting tags that cause XSS vulnerabilities include <SCRIPT>, <OBJECT>, <APPLET> <EMBED> and <FORM>		
4.37	Check the attribute and value elements of the permitted HTML tags. – Reject attributes that are not part of the accepted attributes for the tag. – Reject attributes that do not hold acceptable values. *Example of attributes related to a HTML tag* o *<SCRIPT>: type, language, src* o *<FORM>: action, enctype, method, target*	M	4.36

4.38	Disable HTTP TRACE and HTTP TRACK methods.
	– The TRACE and TRACK are HTTP methods used to debug web server connections. These methods enable attackers to trick a web browser into issuing a TRACE/TRACK request against an arbitrary site and then sending the TRACE/TRACK response to a third party. Attackers can use this to access sensitive information, such as cookies or authentication data, contained in the HTTP headers of the request/response or inquire information from legitimate web users.
	
	To disable TRACE and TRACK in the
	
	Apache Server
	1. httpd.conf
	`...`
	`RewriteEngine On`
	`RewriteCond %{REQUEST_METHOD} ^(TRACE|TRACK)`
	`RewriteRule .* - [F]`
	`...`
	
	2. Virtual host
	`...`
	`<VirtualHost www.example.com>`
	`RewriteEngine On`
	`RewriteCond %{REQUEST_METHOD} ^(TRACE|TRACK)`
	`RewriteRule .* - [F]`
	`</VirtualHost>`
	`...`
	
	IIS
	Settings are determined in the URLScan.ini configuration file.
	The [AllowVerbs] and [DenyVerbs] sections define the HTTP verbs (also known as methods) that URLScan permits. Common HTTP verbs include GET, POST, HEAD, and PUT. There can be additional verbs.
	o If UseAllowVerbs is set to 1, URLScan only permits requests that use the verbs that are listed in the [AllowVerbs] section. A request that does not use one of these verbs is rejected.
	o If UseAllowVerbs is set to 0, URLScan denies requests that use verbs that are explicitly listed in the [DenyVerbs] section. Any requests that use verbs that do not appear in this section are permitted.
		M	

4.39	Ensure that the application does not repeat any encoding-decoding processes at different parts of the application or operating system. – If the data remains encoded, or contains unacceptable characters,	M	

	treat the data as having failed, and deal with it accordingly.		
4.40	Security checks must be performed on decoded data before being stored. For example, ○ Validate if it contains acceptable content, ○ Within maximum and minimum lengths, ○ Correct data type, ○ Does not contain any encoded data, ○ Contains characters of the right format, ○ Check that the encoding is a valid canonical encoding for the symbol it represents.	M	
4.41	Restrict application to use only one canonical format.	M	4.33

Supplementary Information

1. Recommends the use of White Listing to specify acceptable special characters and HTML tags.
2. For message boards, forums, and the like, developers may opt to use BBCode (Bulletin Board Code or Forum Codes). It is designed to provide a safer, easier and more limited way for users to format their messages. The drawback in using BBCode is that, at the time of writing these policies, there is no standard document for BBCode.
3. Apart from the listed precaution measures, it is also necessary that users don't blindly trust links in emails or anywhere other than the main site. It may be wise for an organization to educate its users on how to use the application more safely.

4.1.3 Injection Flaws

Synopsis

Injection attacks are triggered by malicious input being inserted into input fields (form fields, cookies, headers, URL/query strings). The fact that web applications are capable of passing parameters to external systems, operating systems and databases; an attacker can embed malicious input to these parameters which will return undesired or damaging results. To protect against this vulnerability, it is a good practice to use White Listing to create a list of acceptable characters and use regular expressions to match and reject illegal characters or input patterns. This category of vulnerability needs to include all of the controls outlined under Input Validation, Buffer Overflows, Cross Site Scripting, and may be further enhanced with the following controls.

Controls

- Filter and validate all input received by the application
- Include validation methods like data conversion and regular expressions

Implementation Guidelines

(***M***: *mandatory*, ***O***: *Optional*)

Tag	**Policy**	**M/O**	**Related Tag(s)**
4.42	Use regular expressions to match data for authorized or unauthorized content. ***ASP.NET*** An expression is enclosed with a ^ and $ sign. A ^ matches the position at the beginning of the input string and a $ matches the position at the end of the input string. Without these markers, an attacker could affix malicious input to the beginning or end of valid content and bypass validation.	M	4.6, 4.17, 4.29, 4.30

	An example of regular expression for passwords. *(?!^[0-9]*$)(?!^[a-zA-Z]*$)^([a-zA-Z0-9]{8,10})$* A password must be between 8 to 10 characters, contain at least one digit and one alphabetic character, and must not contain special characters.		
4.43	Check the parts of the source code that makes calls to external systems. – Carefully examine each of these calls (e.g. system, exec, fork, Runtime.exec, SQL queries). – Ensure that input data does not contain any specific operating system/SHELL commands. – Avoid accessing external interpreters.	M	
4.44	Handle requests that supply parameters as literal data, rather than potentially executable content.	M	
4.45	Ensure that the web application runs only with right privileges when performing functions across application boundaries. – Disallow users to run the web server as root or access a database as administrator.	M	4.62
4.46	Avoid accepting file names as input. If unavoidable; – Use indexes when trying to retrieve files, – Convert the name to its equivalent canonical form prior to granting access or making security related decisions, – Check whether the file exists within the application's directory hierarchy before returning results.	M	4.15

Supplementary Information
N/A

4.1.3.1 SQL Injection

Synopsis
SQL commands create the interface between a browser and the data stored in a database. Therefore, it is important that a user only gets information he is authorized to obtain. If an application directly accesses a database with user input, an attacker may be able to inject malicious special characters or SQL commands to gain unrestricted access to the application's database. To prevent this, it is essential to apply additional controls, specific to the SQL language, to protect the integrity of databases. This form of validation, together with thorough input validation mechanisms prevents attackers from attempting unauthorized access to an application's database.

Controls
- Filter and validate all user input or data querying a database
- Use prepared statements with parameterized SQL, or
- Use stored procedures with parameterized SQL
- Use least privileged accounts to access databases

Implementation Guidelines
(***M***: *mandatory*, ***O***: *Optional*)

Tag	**Policy**	**M/O**	**Related**

			Tag(s)
4.47	Do not concatenate SQL strings especially for strings that include user input.	M	
4.48	Filter and escape special characters (meta-characters) and reserved words related to the SQL language. *Examples of special characters include apostrophe (‘), command separator (;), comments (--).* *Examples of reserved words include UNION, DROP, DELETE, UPDATE, JOIN.*	M	
4.49	All filtered and validated user inputs must be assigned to parameters. – The parameters are used in prepared statements or stored procedures to query the database. – To enable input values like O’Connor and O’Reilly o Use single quotes for encapsulating data ▪ Split the string on all single quotes, and ▪ Join them using a backslash-escaped single quote *e.g. O’Connor would be processed as ‘O’\’’Connor’* o There should be no space between the words ‘O’ and ‘Connor’. **Note:** *There could be other approaches to handle this kind of input though* *Without these parameters, prepared statements and stored procedures would still be susceptible to SQL injection attacks.*	M	
4.50	Execute stored procedures using a safe interface. *E.g. Callable statements in JDBC or CommandObject in ADO*	M	
4.51	Enforce least privilege policy using access control credentials to; – Distinguish operations that can be executed by different categories of users. – Only give users the rights they need. – Application should not allow users to execute commands as database administrators.	M	4.64
4.52	Assign permissions (write/execute) on stored procedures. – Grant execute permissions to selected stored procedures. – Remove/disable unused stored procedures.	M	
4.53	Avoid disclosing detailed database error information to the user.	M	4.100
4.54	Do not embed database names in application code.	M	4.96
4.55	Validate the number of rows returned from a query retrieving data. – Check that only zero or one record is returned. – Throw an error if multiple rows are retrieved.	M	

Supplementary Information

1. More complex rules are required to prevent SQL Injections in search engines. Further investigation is necessary.
2. Developers must read the database server documentation to identify the characters that need special treatment.

4.2 Access Control

Synopsis

Access control, also termed as authorization, is concerned with protecting data and resources from unauthorized disclosure. It is important in preserving the confidentiality and integrity aspects of an application. Flawed implementations of access control can severely damage the reputation and trust level of users on the application. Most applications control access to data and resources by assigning privileges based on the user's roles and responsibility. Users are categorised into a number of groups or roles that have different abilities or privileges. This information are specified in a web application's access control policy which documents the types of users/groups and their access rights to protected data and resources residing in the application.

Access control mechanisms must be accompanied by a reliable and secure authentication process. Please refer to Section 4.3 for Authentication and Session Management policies.

Controls

- Clearly describe and document a detailed access control policy
- Thoroughly review the code that implements access control logic for correctness
- Plan and perform rigorous testing to detect problems in the access control scheme
- Limit access to protected contents or data via strong authentication and authorization

Implementation Guidelines

(*M*: *mandatory*, *O*: *Optional*)

Tag	Policy	M/O	Related Tag(s)
4.56	The access control policy should contain (but not limited to) the following information: – Identify all possible access points in the application and implement security access control logic at these entries o Which data and resources must be protected o What access requests are required o Who is responsible for granting users access to the protected data and resources – Clearly define the rights and privileges allowed for protected data and/or resources – Define user groups/roles as well as combination of privileges/abilities assigned to these groups/roles to execute operations like read/read&write/execute – Determine the allowed access paths for all groups/user roles to prevent malicious users from navigating to unauthorized data/resources – Ensure access rights of users who have terminated their accounts are immediately changed or removed – Accounts inactive for a specified period of time (i.e. more then two months) should be locked/deactivated	M	
4.57	Propose a formal user registration and de-registration process for granting and revoking access to all critical information systems and services.	O	
4.58	Identify an access control model that well suits the nature and functionality of the application. Some choices include: Role Based Access Control (RBAC) – performs a role check before providing access to sensitive data or resources. Lattice Based Access Control – defines the levels of security that an object may have and that a subject may have access to.	M	
4.59	The method used to identify or reference users, roles, contents, objects	M	4.76,

	or functions (e.g. username, userID, keys) must be strong. – Use strong randomly generated session identifier/token that is not easily guessable (i.e. not containing only timestamps or sequential formulas). – This identifier/token is assigned to a user upon log-on and is associated to the user's identity via a database lookup. *e.g.* ○ *Unacceptable: http://www.mydomain.com/ bad_app?user_id=2123* ○ *Acceptable: http://www.mydomain.com/ good_app?token=36423c633d68564523a* – The supplied identity/session must be further validated to ensure its authenticity with the current user and the attempted access.		4.82
4.60	Separate code implementing access control from application code and keep access control code in a centralised location. *Many access control mechanism are inserted in various locations/pieces of code. Hence, for a large application, the collection of rules becomes so unwieldy that it becomes difficult to understand and maintain.*	M	
4.61	Perform authorization checks on EVERY protected page or URL accessed. – Users should not be able to skip between pages that require security checks before being granted access to certain URLs/files. *Prevents against path traversal attacks*	M	4.88
4.62	Restrict access to administrative pages/sections of a site that should not be accessible to normal users. – Password protect entry to those pages/sections.	M	4.45, 4.73, 4.122
4.63	Do not allow administrator access through a web interface. – If remote administration is required, use VPN/SSL technology to gain access.	O	
4.64	Do not allow application to execute with authorities of an admin, administrator, root, system, super.	M	4.51
4.65	Set the expiry property for protected contents/resources. – Specify allowed access time in session identifiers/tokens. – Use timestamps to expire session identifiers/tokens.	M	4.90, 4.92
4.66	Destroy session identifiers/tokens on the server side when the browser closes or the user logs out. – Prevent browsers from caching information.		4.81, 4.90, 4.92, 4.110
4.67	Protected files (e.g. configuration files, default files, scripts) that are stored locally on application servers should not be publicly accessible. – Specifically mark files using the operating system's permission mechanisms before granting/denying access for read/write/execute operations.	M	4.120, 4.121, 4.123
4.68	Change/verify defaults when using/accessing third party code or applications. – Understand access control assistance provided by the third party/external code/component/application. Does it comply with the application's access control policy? – Determine parts of the access control policy that third party/external code/component/application does not deal with and customized appropriate access control logic for these parts.	M	4.125

4.69	Limit file permissions on files accessible via the web.		

Supplementary Information

1. Establish a classification scheme for all data and resources (e.g., public, confidential, secret, top secret).

4.3 Authentication and Session Management

Synopsis

Authentication and authorization are two interrelated concepts in security. While authorization is concerned with determining if an authenticated user has the right access to requested resources, authentication is the ability to determine whether the user is the person he claims to be. Authentication is usually performed using credentials like usernames, userID, passwords, challenge response. An incorrect or incomplete implementation of authentication can result in breaching the privacy and integrity of data belonging to an individual or organization. Hence, authentication can be seen as the single most important requirement for safeguarding privacy in a networked world.

Session management, on the other hand, is required to keep track of the stream of requests made by each user. Session identifiers/tokens are presented to the server as hidden fields, appended to URLs or stored in cookies. Typically the information in sessions is used to enforce rules about page access, record purchases (shopping basket items), track site history of previously viewed contents.

Together, authentication and session management are responsible in handling and managing active sessions in web applications.

Controls

- Choose the most appropriate form of authentication per the application's risk
- Re-authenticate users for high value transactions or information
- Use strong session identifiers and protect them throughout their lifecycle

Implementation Guidelines

(*M: mandatory*, *O: Optional*)

Tag	**Policy**	**M/O**	**Related Tag(s)**
General Authentication and Session Management Policies			
4.70	All authentication credentials and session identifiers/tokens must be protected with SSL/TLS technology during transmission. *Ensures that all session and data exchanged between the server and client remain confidential and tamper-proof during transit.*	M	4.1
4.71	Authentication and session data should never be submitted visibly in URLs. – Always opt for the POST method. – Encrypt or hash authentication and session information transmitted. – Parameters must be accompanied by a valid session token.	M	
4.72	Divide the application into two parts (on separate servers if it's for a large application); one for HTTP requests and the other for HTTPS requests.	O	4.2
4.73	Configure the server to require authentication at the directory level for all files within the directory.	M	4.122, 4.45, 4.62

4.74	Each component (internal/external to the code/application) accessing the application should authenticate itself to any other component it is interacting with. *This, however, may impact the performance of the application.*		
4.75	Never rely on information from the HTTP Header for authentication or authorization decisions.	M	4.12
Authentication			
4.76	Ensure strong authentication mechanisms for highly sensitive information. – Strong authentication (such as tokens, certificates, transaction signing) can provide higher level security for legally compelled information such as health records and government information. – Combine the use of credentials like username password with a possessed item/device. Also known as two-factor authentication.	M	4.59, 4.82
4.77	Implement a delay between submitted credentials and success or failure report. – Force time delay between incorrect logon attempts before locking access. *Prevents brute force or application DoS attacks*		4.19
4.78	At each logon session, users should be informed of – the date/time of their last successful login, and – the number of failed access attempts to their account since then.	M	
4.79	If a user has not logged in for a specified period of time, delete or deactivate the associated account unless given upfront notice.	M	
4.80	Avoid the use of "Remember Me", "Remember My Password", "Sign Me in Automatically" options.	M	

4.81	Authentication pages should be marked with the "no cache" tag to prevent the use of back/refresh button in the browser.
	– Many browsers now support the autocomplete=false flag to prevent storing of credentials in autocomplete caches.
	
	Example,
	`<form … AUTOCOMPLETE="off">  - for all form fields`
	`<input … AUTOCOMPLETE="off"> - for one input field`
	
	Prevents the resubmission of previously typed credentials.
		M	4.66, 4.110

Session Management			
4.82	Create random/unique session identifiers/tokens using a carefully chosen/designed algorithm. – The session identifier/token issued must be unpredictable (i.e. not sequential, does not use an easily recognizable pattern). o Use strong (pseudo) random numbers o Ensure sufficient randomness o Ensure sufficient length – The algorithm used should involve a lot of permutations that makes it hard to be computed	M	4.59, 4.76
4.83	Ensure that users get a new session identifier/token with each visit/revisit to the site. – Use different session identifiers when shifting between secure and insecure contents (e.g. authenticated vs. non-authenticated, http vs. https).		

	– Verify that all subsequent requests to restricted contents and/or resources are received from that same user's host origin until the user logs out.		
4.84	Allow only one session per user at a time. – If a new session is started for the same user, implement logout functionality. – Block access to users attempting to access more than one protected information at any one time. – Use one session identifier/token to reference properties stored in a server-side cache. – Limit each sessions to a set of IP addresses.	M	
4.85	Session identifiers/tokens must always refer to information stored on the server.	M	4.20
4.86	Do not store session status or other critical session information in cookies, URL/query strings or hidden fields. – If session information must be stored on the client, then it should be o encrypted/hashed, o applied with integrity control mechanism like checksum, digital signature, HMAC, etc.	M	4.11
4.87	If the application uses session identifiers/tokens in query strings/cookies/hidden fields, use a different pattern for each of the identifier/token used.	O	
4.88	Restrict the sequence in which users can view pages. – Compare the last visited page against the one a user should have come from. o Store the last visited page in session variables on the application server. o Use cookies (temporary/persistent) to store last visited pages or identifiers. Temporary cookies expire when the browser closes.	M	4.61
4.89	Important things to remember when using Cookies; – If possible, avoid the use of permanent cookies – Restrict the use of cookies to particular sites/sections of a site – Don't rely on the cookie's own expiry date – Set cookies to expire automatically – Encrypt cookies to prevent tampering o Hash the cookie and compare the hash values with the hash value stored in the server. o Perform symmetric encryption This method however will be invalidated once the server is compromised. Then, a new key must be generated to continue encryption. – Include corresponding path restrictions in the cookie path option – Set the Secure and HttpOnly properties on the cookie – Keep a reference in the cookie to a location on the server where the information can be verified	M	4.86
4.90	Expire/destroy/lock session identifiers/tokens when the browser/application is closed or user remains idle for a certain amount of time or implement session logout mechanisms. Points that follow: – Delete/overwrite the browser's session cookie (persistent/non-persistent). – Allow users to log out and then clear the session.	M	4.19, 4.65, 4.66

	○ Invalidate current as well as all other sessions that the users have failed to log out from at both the client and the server. – Session tokens should be expired on the server. – Include time-out session identifiers to prevent reuse after a pre-determined period of time.		
4.91	Do not allow users to reactivate session identifiers/token if the valid session identifier/token has already expired.	M	
4.92	Expire URLs at specific points in time to limit access to protected contents without requiring additional authentication process. – In ASP.NET, this is done using the *ExpireTime* property of a *SecureQueryString* instance to be set.	M	4.65, 4.66

Supplementary Information

1. The above mentioned policies not only protect against Broken Authentication and Sesssion Management vulnerabilities, but also Cross Site Scripting, Injection Flaws, and Insecure Storage.
2. Code review and penetration testing can be the additional steps used to diagnose authentication and session management problems.
3. Examples of authentication mechanisms include
    - username and passwords/HTTP Digest Authentication – suitable for low value systems
    - challenge response – suitable medium value systems
    - transaction signing – suitable for high value systems

4.4 Error Handling and Logging

Synopsis

An error is the occurrence of an incorrect result that may be due to an invalid operation caused by the user, or system errors resulting from illegal or illogical operations that fail outside the scope of the application. Another important concept in this category is exception. Exception handling refers to the ability of an application to accept and deal with exceptional failures in the most secure manner possible. An unhandled or improperly handled exception can cause a server to crash or may even result in denial of service for its users.

Poorly constructed error and exception handling messages can feed malicious users with information about the underlying structure (i.e. code, databases, server type, and operating system) of the application. This information can then be used to reverse engineer and eventually exploit the application.

To monitor an application from undesired mishandling or breaches, logging mechanisms must be implemented. Periodic auditing process on the logs helps to identify if there had been any recent malicious attempts on the application. Consequently, necessary actions can be taken to prevent the application from being compromised. Without appropriate logging, attackers can deny performing malicious operations or exploit and application without leaving a trace. Logging also helps developers to debug errors or exceptions occurred in the application. Therefore, it usually contains a lot of information that reveals the internal structure of the application. Logging information must be protected from tampering and unauthorized access.

Controls

- Ensure that failure and exceptional conditions are dealt in a secure manner
- Ensure proper formulation of error and exception messages to end users
- Periodically audit log files for misuse or leaked information

Implementation Guidelines

(*M: mandatory*, *O: Optional*)

Tag	Policy	M/O	Related Tag(s)
General Error and Exception Handling Policies			
4.93	Use try/catch/finally blocks around code. – Try statement generates the exception. – Catch statement handles the exception. o Centralize catch statements in one location. o Always order the catch clauses from most specific to least specific. In this way, the most specific exceptions are handled first. – Finally statement closes or removes resources. – Avoid writing try/catch/finally blocks with an empty catch block. – Even if an exception is not anticipated, it is safe programming practice to add some code to signal an exceptional event in any case.	M	
4.94	Make sure the system is not left in an inconsistent or insecure fail-open state. – Use catch/finally blocks to cleanup or release resources. o If catch blocks are used to release claimed resources, make sure it releases resources in the try block as well. o Finally blocks for unifies and simplifies the release of resources.	M	
4.95	Correctly formulate error messages for all expected/unexpected events. – Confirm if its level of detail is sufficient to serve an average user. *For example, if application error is due to invalid user input, such as an invalid password, then the application should present a generic and non-distinct error message like the username/password combination does not match.*	M	4.100, 4.127
4.96	Remove sensitive information/comment embedded within the code. – Review any comments contained in the source code. – Each programming languages have their own commenting style o // o /*…..*/ o <!--- These comments shouldn't be echoed to the client.	M	4.54
4.97	Ensure that the application has a "safe-mode" for handling unexpected events. – If all else fails, log the user out and terminate the session.		
Errors and Exceptions in Application Server, Web Server, Database Server			
4.98	Configure servers to *NOT* show default/verbose error messages. – Redirect errors to a generic page. o Use the global error handler at the page or application level to catche and route errors/exceptions to a generic page. – Allow the server to deliver custom pages. – Default error messages or debug settings must be available ONLY to the administrator group of the server. – Consult the server's documentation to lock-down or turn-off specific default settings.	M	4.99, 4.100

	– This includes HTTP status response codes. *Hence, in unexpected situations the users will not see detail information about the error/exception*		
4.99	Do not reveal any debugging information like ○ server error/server banners, ○ print stack trace, ○ name or type of the application/web/database server. *i.e. Apache (version 1.3x and 2.0x)*	M	4.98, 4.127
4.100	The application should provide a generic message containing a tracking number. The tracking number is used to help developers resolve the error/exception. – All error/exception messages must be well documented in a separate file or error/exception resource library, unique for the application. *e.g.* *Tracking Number: JK01* *ErrorDocument JK01 /webserver_errors/server_error500.txt*	M	4.53, 4.95, 4.127
4.101	Ensure that the application does not return different errors messages for – files that do not exist, – files that exist but perhaps denies read/write/execute access, – directories that do not exist, and – directories that are not accessible. Ensure that secure paths that have multiple outcomes return similar or identical error messages.	M	4.127
Logging			
4.102	Create application level logs. – Do not rely solely on web server logs that contain mostly HTTP-level information. – All critical application level operations should be logged . ○ *E.g. Authentication (including failed authentication) attempts, application behavior, user actions.* *Application logs can reveal application-level attacks*	M	
4.103	Data that should be logged include;	M	

Reference Number	Date	Time
Session/Token ID	Request Method	HTTP/HTTPS
Server Address	Server Port	User's Browser Type
URL that caused the error/exception	HTTP headers	Category
Source	File	Line & Column
Message/Description/ Message Tracking Number		Log Level DEBUG/INFO/ERROR/F ATAL

	Information logged depends on the application and the compliance of the application with certain constitutions.		
4.104	Do not keep log files in default location folders. – Save logs in location only available to the administrators and auditors.	M	4.120

	○ Secure log files with proper access control mechanisms. – The location should not be within the same disk/system partitions that is running the application. Otherwise it may be subject to application DoS attacks.		
4.105	Always archive and maintain a backup copy of log files. – Archive/backup log files at pre-determined intervals. – If possible, store back-up copies in different locations/servers.	M	
4.106	All output, return code and errors generated must be checked and logged to ensure that the expected processing actually occurred. *At a minimum, this would determine if something fraudulent had occurred. Otherwise, an attack may success undetected.*	M	
4.107	The unique session/token/user identifiers can be used to link users for their actions (i.e. malicious or otherwise).	M	
4.108	Ensure that logs cannot be overwritten or tampered by local/remote users. – Set a large size for the log file and once it nears its limit, create a new log file.	M	4.116
4.109	Perform periodic review of log files (daily/weekly/bi-weekly) to detect any faults or suspicious activity.	M	

Supplementary Information

1. Additionally, use log monitors at all levels of the application to scan log files in real time or at given intervals. The logging mechanism should have the capability of generating reports based on the logging data and alerting application/network administrators when critical events occur.
2. All logging components should be synced with a timeserver so all logging are done without latency. The time server should be hardened and not used for other services.

4.5 Secure Storage

Synopsis

A secure storage mechanism is important to maintain the integrity of the application and the data stored within it. Data in web applications are often held persistently in databases and file systems, and non-persistently in sessions (cookies/hidden fields). Depending on the functionality of the application, sensitive data can range from proprietary information like passwords, account statements, medical history, to top secret information help by government or military groups. To prevent against data leakage applications should limit the amount of data retrieved and the amount of sensitive data stored by the application. The use of encryption along with strong key protection mechanisms can afford secure storage within web applications.

Controls

- Persistent stores should store sensitive information in encrypted formats
- Protect all resources from unauthorized access, disclosure, modification, or destruction
- Ensure that cryptography is safely used to protect confidentiality and integrity of data

Implementation Guidelines

(***M***: *mandatory*, ***O***: *Optional*)

Tag	**Policy**	**M/O**	**Related Tag(s)**
4.110	Program browser's cache to be automatically deleted/cleared when the user logs out or the browser window is closed. – New browsers support caching control using these headers	M	4.81, 4.66, 4.90

	○ Cache-Control: no-store, no-cache, must-revalidate ○ Cache-Control: post-check=0, pre-check=0		
4.111	Stick to existing well known standard cryptographic algorithms. – Do not use own/proprietary algorithms.	M	4.11, 4.22, 4.86, 4.89, 4.112
4.112	When using encryption that generates and uses keys, ○ Use strong random key generation functions, ○ Check if key size is sufficient for the application, ○ Store keys in restricted locations with restricted access control, ○ Expire keys regularly,	M	4.22, 4.111

Supplementary Information

1. For storing credit card information, the application must comply with the guidelines of the Payment Card Industry (PCI) Data Security Standard

4.6 Application Denial of Service

Synopsis

In this category of vulnerability, attackers aspire to make an application inaccessible by disrupting its service. This is done by crashing the application or its database such that no other user can use the application. Application denial of service is closely linked with Input Validation, Buffer Overflows, Injection Flaws, Access Control and Error Handling and Logging vulnerabilities. The use of sensitive information, complex calculations, heavy-duty searches should be reserved for authenticated and authorized users only. Every feature and functionality of an application should be engineered to perform fast, few and secure database queries, and avoid exposing large files or unique links to prevent denial of service attacks.

Controls

- Ensure proper input validation and access control mechanisms
- Properly handle errors and exceptions so that the application is not affected at any one point
- Avoid unnecessary accesses to databases or other expensive resources

Implementation Guidelines

(*M: mandatory*, *O: Optional*)

Tag	Policy	M/O	Related Tag(s)
4.113	All connections to protected contents or data must have timeouts. – Limit the resources allocated to a user at any one point. – Set timeouts for all user sessions.	M	4.65, 4.90, 4.92
4.114	Avoid high CPU consuming operations like large computational operations that take a long time to complete. – Split operations into chunks. – Prepare for performance peaks, ○ Load balancing ○ Caching	M	4.94
4.115	Limit the amount of load a particular user can put on the application. – Confine the length of incoming data to particular limits. – Handle one request per user at a time by synchronizing the user's session.	M	4.19, 4.77, 4.84

	– Drop any requests that you are currently processing when another request from the same user arrives.		
4.116	Increase log file size to a maximum value that is unlikely reached. – Add a function that can monitor the log file size. – Deploy a monitoring tool to monitor logging activity and issue an alert for attacks/abnormal activities.	M	4.108

Supplementary Information

1. Perform load testing on the application using tools that can generate high web traffic to test how the application performs under heavy loads.

4.7 Configuration Management

Synopsis

Configuration management is essential for web developers and administrators to update web site content, modify or repair code, change configuration parameters, perform routine maintenance, and various other related tasks. It also addresses the association between documents, revision history, and the persons or group responsible for the maintenance of documents. A quality review process is useful to ensure proper accordance of configuration management rules. Furthermore, a manual review/check of the configuration document/plan should be performed regularly to ensure that it has been kept up to date and is in consistent state.

Controls

- Create and maintain separate configuration guidelines for all servers (web server, application server and database server)
- Store all configuration information in secure locations only accessible to authorised users
- Maintain an updated inventory of applications/files/databases that reside on the servers
- Perform regular vulnerability scanning from both internal and external perspectives
- Provide periodic status reports documenting overall security status

Implementation Guidelines

(*M: mandatory*, *O: Optional*)

Tag	Policy	M/O	Related Tag(s)
4.117	Establish a unique naming convention for all directories, files, and databases. – Use correct extensions for files containing code.	M	
4.118	Block URL's that include dot-dot and backslash characters. – Remove all double dots (\..\..\..) from GET and POST parameters. – Remove sequence of dots represented in Unicode/hexadecimal or other representations. *This would disable directory traversal attacks.*	M	4.29, 4.42
4.119	Web servers use a "permission by directory" model where files are accessible according to the directories they are in. – Create a boundary about what content can be viewed and what cannot. o Create the /include directory out of the web document root to a directory where an unauthorized web user cannot access but the designated web server can. Hence, library files and header code cannot be executed or read from a browser.	M	

	○ Place all data files in a directory outside of both the /public_html and /cgi-bin directories so that the files cannot be read or executed from a browser. Every file in the /public_html directory can be accessed and /cgi-bin directory can be executed if the URL is known/guessed. – Configure the server to take the deny-by-default stance. (specify only what is allowed and prohibit everything else)		
4.120	Don't store sensitive and publicly accessible data on the same directories or servers. – Keep separate maintenance and publication areas of all documents – Unknown/sensitive files (with extensions .asa, .inc, etc) should not be returned to the client. – Directories without an index file should not have files listed.	M	4.67, 4.104
4.121	Remove/disable any sensitive or non-essential files/applications/databases from easily accessible directories. – This includes default, backup, scripts, configuration files, web pages, data files, log files, old and unreferenced files. – Use scanning tools such as Nikto, Paros, AppScan, WebInspect, etc to locate files. – Locate and remove/relocate files that haven't been accessed in an extended period of time (using the last access timestamps). – Keep custom configuration files outside the directory that doesn't have web access.	M	4.67
4.122	Make sure passwords are enabled for sensitive areas and administration functions.	M	4.62, 4.73

4.123	If certain files need to remain in specific locations on the web server, configure Apache and IIS to password-protect and/or IP restrict the resources.
	For example, suppose the directory "/var/www/html/private" must be protected
	Apache:
	1. Edit *httpd.conf* to include the following directive:
	`<Directory "/var/www/html/private">`
	`AllowOverride all`
	`</Directory>`
	2. Create a file called *.htaccess* inside */var/www/html/private*. It should contain something like the following:
	`AuthName "Private Directory"`
	`AuthType Basic`
	`AuthUserFile /var/www/html/private/.htpasswd`
	`Require user privateuser`
	3. Create a file called *.htpasswd* in */var/www/html/private* which will contain the allowed username and password in encrypted form
	`htpasswd -c /var/www/html/private/.htpasswd privateuser`
	
	The *.htaccess* file contains directives that apply to the files in the same directory (including subdirectories) as the *.htaccess* file.
	
	IIS:
	Using tools like IISPassword or IISProtect that are able to password protects all file types. They can also be integrated with Microsoft Management Console which provides an easy to use interface that allows you to add user accounts and select the folder(s) to be protected.
		M	4.67

4.124	Turn off Apache "Directory Indexing" or IIS "Directory Browsing" in	M	

	the web server configuration.
	
	Apache: In the mod_autoindex file,
	1. To turn on automatic directory indexing, find the Options directive that applies to the directory and add the Indexes key word.
	`<Directory /path/to/directory>`
	`Options +Indexes`
	`</Directory>`
	2. To turn off automatic directory indexing, remove the Indexes keyword form the appropriate Options directive.
	`<Directory /path/to/directory>`
	`Options -Indexes`
	`</Directory>`
	
	IIS:
	To turn off directory browsing, open up the IIS Admin Tool (inetmgr) and go to the properties attribute of the application, vdir or directory. There should be a "allow directory browsing" checkbox on the tab that can unchecked.
	
	This feature is commonly used to create web pages for directories without index files (index.html, default.htm, etc). This feature can aid an attacker in locating sensitive files on the web server.		

4.125	Do not accept server defaults without analysis. – If the server is within a shared environment (multiple services on the same server, or multiple servers performing specific tasks for the application), do not allow sharing of directories. – Verify that permissions have been set up correctly.	M	4.68
4.126	Incorporate explicit version control mechanisms.	M	
4.127	Display generic error messages irrespective of how the application deals with errors and exceptions. – Prevent application-level trace information from being displayed to the end user.	M	4.95, 4.99, 4.100, 4.102
4.128	Limit executable files to specific directories and make sure their source code can't be downloaded. – Some file extensions when accessed are either displayed or downloaded by the browser. Files with these extensions must be checked to verify that they are indeed suppose to be served and do not contain sensitive information. o Compressed archive files: .zip, .tar, .gz, .tgz, .rar, etc o Source code files: .java, .aspx, etc o Text files: .txt o Pdf documents: .pdf o Office documents: .doc, .rtf, .xls, .ppt, etc o Others: .bak, .old, etc	M	
4.129	Ensure application's temporary files are not accessible from web root	M	

Supplementary Information

1. Document and ensure strict adherence to configuration management process
2. Constantly update the servers and its operating system with latest virus scanners and detectors.

Summary

The policies and guidelines presented in this chapter are aimed for developers to implement web applications using secure coding. It takes into account the fundamental coding requirements to circumvent the threats and vulnerabilities discussed in the preceding chapters. Naturally, a point that remains ambiguous is how complete and effective are these policies. The policies cover at large the known attacks, threats and vulnerabilities today's web applications face. With time, more sophisticated attacks will surface requiring stronger and more robust defensive mechanisms in the code. The above listed policies and guidelines can be easily extended to include more policies and guidelines as well as to add more threat and vulnerability categories.

Most of the policies and guidelines are tagged as mandatory (M). It may be infeasible to materialize all of the mandatory policies, due to development time and other constraints. However, the reason for it being mandatory is so that web developers and potentially the application architects and designers can consider if the mandated policies will be useful in their intended application. Optional tags on the other hand are dependent on the functionality of the web application and can be incorporated as an additional control.

Some of the suggested methods shown may have a more improved or smarter way to code depending on the knowledge and experience of web developers. The suggested methods serve only as a means to demonstrate the possible implementation of the policy.

A reference card has been developed to provide a comprehensive overview of the important elements in this chapter. Please refer to Appendix D.

6 Conclusion

The primary objective of this thesis was to formulate a set of coding policies and guidelines to assist web developers in incorporating secure coding. In order to achieve this goal, we first studied the development lifecycle of web applications to see how security initiatives can be applied in each of its phases. We walked through examples of writing security requirements, building use and misuse cases, discussing various testing strategies, to name a few. The phases within the development lifecycle are interrelated and are revisited each time there is a new requirement or discovery in any of the phases. Hence, a close liaison among requirements engineers, architects, designers, developers, and testers is essential as their experience and skill sets are largely complementary. Chapter 1 asserts that secure web applications are attainable only if security gets a position in all stages of the development lifecycle.

In Chapter 2, we took a closer look at existing web application vulnerabilities as reported in many of the articles, conferences and organizations dedicated towards web application security. Some of the reputable articles originate from organizations like the Open Web Application Security Projects (OWASP), Web Application Security Consortium, Fortify Software, Inc., SPI Dynamics, and Gartner Research.

The most occurring web application vulnerabilities are *unvalidated input*, *buffer overflows*, *cross site scripting*, *injection flaws*, *SQL injection, broken access control*, *broken authentication and session management*, *improper error handling and logging*, *insecure storage*, *application denial of service*, and *insecure configuration management.* We saw many of the attacking approaches and methods used by attackers in breaching the security of web applications. Analysis showed that the attacks are mostly possible because of implementation flaws. A simple flaw like forgetting to perform input validation can lead to more than one type of attack. For instance, an unvalidated input can cause an injection flaw that leads to a buffer overflow that in turn could eventually crash the application causing an application denial of service.

The focus on destruction methods in Chapter 2 demonstrated that every line, function or class of code written must consider all possible threats surrounding the application. A small mistake or misstep can be very dangerous. Chapter 2 showed how significant coding is with regard to web application security.

Having studied the various threats and vulnerabilities as well as some of the attacking strategies, we took a look at the available prevention mechanisms in two of the most widely used web development languages, i.e. Java and ASP.NET. We learned about the existing libraries, classes, and frameworks in Java and .NET that can be used to avoid many of the web application vulnerabilities. Both languages cater for a wide range of mechanisms that offer security based solutions for web applications.

Java and .NET offer specific APIs and frameworks to address each of the discussed vulnerabilities. The mentioned APIs and framework may require the inclusion of other APIs and frameworks to develop a more complete implementation. However, Java and .NET treat vulnerabilities related to insecure configuration management and application denial of service rather differently. Issues of insecure configuration management are not directly related to the application's implementation. To avoid insecure configuration management vulnerabilities, developers can adopt one of the many configuration management tools. In this thesis, we suggested the SVNKit for Java applications and Microsoft's Management Console (MMC) snap-in for .NET applications. On the other hand, both Java and .NET do not have specific APIs and frameworks to prevent application denial of service (DoS) attacks. Generally, there are no perfect measures to protect against application DoS. Since most application DoS are caused by weak mechanisms in the other vulnerability categories, by ensuring adequate prevention methods in those categories, the occurrence of application denial of service can be controlled. The properties of the Java Virtual Machine (JVM) in Java and the Common Language Runtime (CLR) in .NET automatically defend buffer overflow vulnerabilities. However, it is still crucial for developers to take additional precautionary measures and not to depend on the JVM and CLR. This will further ensure that the program is stronger against buffer overflows and is unlikely to miss any important checks.

Even with the large collection of APIs, tools and frameworks, web applications still face many threats and vulnerabilities. Analysis show that possible reasons why Java developers are not aware or do not use many of these built-in mechanisms are because it is difficult to find information as information is widespread without clear cross references or links to related documents. As for .NET developer, one reason could be because much of its security controls seem to be only applicable for applications within the .NET environment.

On the other hand, some API documentation incorrectly gives the impression that security is taken care of. For example, the J2SE API Specification version 1.4.1 for Java Locale setting, states that the *getLanguage* method returns either an empty string or a two letter language code. In reality, the *getLanguage* method returns whatever raw user input passed to the Locale when the instance is created. This could result in an attacker manipulating the Accept-Language header in the HTTP Header request. Hence, developers should not blindly trust any documentation without performing further analysis, tests or validation checks.

Comparison in terms of competency between the mechanisms provided by the two languages is currently difficult to judge objectively as it requires a more in-depth analysis and probably testing of real applications.

Chapter 3 gave good insights on the available prevention mechanisms for both Java and .NET. This together with results of the preceding chapters led to the formulation of secure coding policies and guidelines, which is the major contribution of this thesis. The coding policies are written based on the current known attacks and vulnerabilities with additional preventative measures for other likely attacks. A reference card has also been developed to provide a comprehensive overview of the important elements in secure coding policies and guidelines.

Security in web applications is a continually evolving process. As attacking strategies change, the coding policies and guidelines will also need to undergo updates and changes. Hence, with time, the policies may require some revisions. Nonetheless, the format in which the policies and guidelines are written makes them easily extensible to include more prevention measures as well as to add more threat and vulnerability categories.

The coding policies presented in this thesis do not contradict each other. Most of the policies address the specific threats and vulnerabilities affecting today's web applications. However, it may not be feasible to materialize all of the listed policies due to possible tradeoffs especially with respect to performance and usability of the application. For example, expiring URLs at specific points in time to limit access may affect the usability of the application. Users will need to re-authenticate themselves in order to use the contents of the URL. Moreover, if session information is used on the client side, the encryption of sensitive information may impact the speed of the application response time. An important question is which of these tradeoffs can be accommodated at the expense of security.

A point that remains ambiguous is how complete and effective these coding policies and guidelines are. Although the policies have been based on extensive research, this can only be measured once they have been applied in constructing web applications along with performing rigorous testing to check for existing or newer security gaps.

Much of the research on this topic showed that many of the vulnerabilities are caused by simple programming mistakes. Hence, apart from the coding policies and guidelines, developers must also be mindful not to develop applications hastily. The quick reference card designed makes it possible for developers to write secure code in a more structured manner while being able to seek information fast.

It is also important to ensure that there is a clear interface between code written to address security and code written to address the functionality of the application. This is important because when new or improved requirements or secure coding policies are introduced, the application is able to undergo changes without introducing a chain of reaction changes. More importantly, this will help preserve both the security and functionality aspects of the web application.

The activities in the application development lifecycle, the examination of the various threats and vulnerabilities in existing web application plus the research of available prevention mechanisms in two widely used web development languages ultimately led to the formulation of secure coding policies and guidelines. We hope that these coding policies and guidelines will be a useful assistance to web development teams in practicing secure coding when developing web applications.

REFERENCES

[1] Security Innovation, Inc. *Application Security by Design.* February 2006.

[2] Pescatore, J., Lanowitz, T. *Sanctum Buy Shows Security Is Key to Application Development*. Gartner Research, 30 July 2004.

[3] Davis, N. *Secure Software Development Life Cycle Processes*. Technical Note, Carnegie Mellon University, July 2006.

[4] Barnum, S. and Sethi, A. *Attack Pattern Usage*. Cigital, Inc. November 2006.

[5] Kotonya, G., Sommerville, I. *Requirements Engineering: Process and Techniques.* John Wily & Sons. May 2004.

[6] McGraw, G. *Software Security: Building Security In*. Addison-Wesley, 2006.

[7] Stiennon, R. *Magic Quadrant for Enterprise Firewalls*. Gartner Research, June 2003.

[8] Lassila, O., van Harmelen, F., Horrocks, I., Hendler, J., McGuinness, D.L. *The Semantic Web and its Language.* Intelligent Systems and Their Applications, IEEE, Volume 15, Issue 6, Nov/Dec 2000. Page(s):67 – 73.

[9] Taccolini, M. *Security Issues with Distributed Web Applications.*2002.

[10] Wesley, D. *Microsoft Application Center 2000 Resource Kit: Chapter 1 - Scaling Business Web Sites with Application Center.* 2000.

[11] Sima, C. *The Latest in Internet Attacks: Web Application Worms.* SPI Dynamics, Inc. 2005.

[12] Andrews, M., Whittaker, J.A. *How to Break Web Software: Functional and Security Testing of Web Applications and Web Services.* Addison-Wesley. 2006.

[13] Grossman, J. *Myth-Busting Web Application Buffer Overflows.* Whitehat Security Reports. WhiteHat Security, Inc. 2006.

[14] Huseby, S.H.*Innocent Code: A Security Wake-Up Call for Web Programmers.* John Wiley & Sons, Ltd. 2004.

[15] Sima, C. *Security at the Next Level: Are your web applications vulnerable?* SPI Dynamics, Inc. 2005.

[16] National Security Telecommunications and Information Systems Security Committee. *National Information Systems Security (INFOSEC) Glossary.* NSTISSI No. 4009. September 2000.

[17] Carlsson, B., Baca, D. *Software security analysis - execution phase audit.* Proceedings of the 31st EUROMICRO Conference on Software Engineering and Advanced Applications (2005) 240-247.

[18] Wikipedia. *Stack In Computing.*

[19] Skoudis E., Liston, T. *Counter Hack Reloaded: A Step-by-Step Guide to Computer Attacks and Effective Defenses.* Chapter 7, Second Edition, Prentice Hall. December, 2005.

[20] Alcorn, W. *The Cross-site Scripting Virus*. Whitepaper at BindShell.Net. October 2005.

[21] Cenzic, Inc. Top 10 vulnerabilities in Web applications in Q1 2007. 22 May 2007.

[22] Friedl, S. *SQL Injection Attacks by Example*. Unixwiz.net Tech Tips. July 2007.

[23] Berg, A. *Improper error handling.* Information Security Magazine. 26 August 2005.

[24] Brenner, B. *Web applications caught in a storm of attacks*. Technical Report. Information Security Magazine. 18 July 2006.

[25] Open Web Application Security Project (OWASP). *The Ten Most Critical Web Application Security Vulnerabilities.* 2007.

[26] Microsoft Office Communicator Web Access Security Guide. *Identifying Possible Security Threats*. Microsoft TechNet. April 14, 2006

[27] Ollmann, G. *HTML Code Injection and Cross-site scripting*. Technicalinfo.net. 21 May 2007.

[28] SearchSecurity.com staff. *Security Bytes: Injection flaw in popular browsers*. Information Security Magazine. 9 December 2004.

[29] Curphey, M., Araujo, R. *Do Configuration Management During Design & Development*. Software Mag.com. October 2005.

[30] Enrigh, S. *Handling Java Web Application Input*. Sun Microsystems, Inc. java.net. 2005.

[31] Klein, A. *DOM Based Cross Site Scripting or XSS of the Third Kind. A look at an overlooked flavor of XSS*. Web Application Security Consortium. July 2005.

[32] SearchSecurity.com. *Web application attacks Learning Guide.* Information Security Magazine. 11 May 2006.

[33] Lavenhar, S. Code Analysis. Cigital, Inc. 22 February 2006.

[34] Fong, D. *Build extra secure Web applications*: *A new protection framework helps you guard against action and data tampering.* developerWorks, IBM's resource for developers. 1 November 2005

[35] Web Application Security Consortium. *Threat Classification: OS Commanding*. 2005.

[36] Curphey, M., Araujo, R. *Getting it Right: Error Handling and Exception Management.* Software Mag.com. March 2006.

[37] Radhakrishnan, S. *Configuration Management in Java EE Applications Using Subversion.* O'reilly OnJava.com. 3 May 2006.

[38] Mahmoud, Q. H. *Web Application Development with JSP and XML, Part I: Fast Track JSP.* Sun Developer Network (SDN), June 2001.

[39] Bergsten, H. *JavaServer Pages.* 3rd Edition. O'Reilly Media, Inc. December 2003.

[40] Johnson, M. *A beginner's guide to Enterprise JavaBeans: An introductory overview of the Java server-side application component standard.* JavaWorld.com. 1998.

[41] Dubin, J. *Java security: Is it getting worse?* Information Security Magazine. 12 July 2007

[42] Mesbah, A., van Deursen, A. *Crosscutting Concerns in J2EE Applications.* Center of Computer Science and Engineering. Amsterdam.

[43] Sindre, G., Opdahl, A.L. *Templates for Misuse Case Description.* Department of Computer and Information Science, Norwegian University of Science and Technology. CiteSeer.IST. 2001.

[44] McGraw, G. and Felten, E. *Securing Java.* John Wiley & Sons, Inc. 1999.

[45] Hoglund, G., Butler, J. *Rootkits:Subverting the Windows Kernel..* Addison Wesley Professional. July 2005.

[46] Bentley, C., Watterson, S.A., Lowenthal, D.K., Rountree, B. *Implicit array bounds checking on 64-bit Architectures.* ACM Transactions on Architecture and Code Optimization (TACO). Volume 3, Issue 4. December 2006. Pages: 502 - 527

[47] Sterbenz, A., Lai, C. *Secure Coding Antipatterns: Avoiding Vulnerabilities.* JavaOne Conference, Sun Microsystem, Inc. 2006.

[48] Mitchell, S. *Sams Teach Yourself ASP.Net in 24 Hours.* Sams Publishing. May 2003.

[49] Meier, J.D., Mackman, A., Dunner, M., Vasireddy, S. *Building Secure ASP.NET Applications: Authentication, Authorization, and Secure Communication.* Microsoft Corporation. January 2006.

[50] Gabhart, K. *J2EE pathfinder: Filtering with Java Servlets 2.4. Viewing, extracting, and manipulating HTTP data with Servlet filters.*developerWorks, IBM's resource for developers. 27 January 2004.

[51] Ford, S., Wells, D., Wells, N. *Web Programming Languages*. WebApps Magazine. Vol. 1, No. 1, January/February 1997.

[52] Volkheimer, J. *Introduction to Servlets, JSP, and Servlet Engines.*DevCentral, Interface Technologies, Inc. 2001.

[53] Singer, M. *J2SE 1.5: A Tiger By the Tail.* Internetnews.com. 28 June 2004.

[54] Nagel, C. *Enterprise Services with the .NET Framework: Developing Distributed Business Solutions with .NET Enterprise Services.* Addison Wesley Professional. June 2005.

[55] Piliptchouk, D. *Java vs. .NET Security*. O'Reilly Media, Inc. June 2004.

[56] van Solingen, R., Berghout, E. The Goal/Question/Metric Method: A practical guide for quality improvement of software development. McGraw-Hill Publishers, 1999.

[57] McDermott, J., Fox, C. *Using Abuse Case Models for Security Requirements Analysi*s. Proceedings of the 15th Annual Computer Security Applications Conference. 1999.

[58] Barnum, S. *Attack Patterns: Knowing Your Enemy in Order to Defeat Them.* Cigital, Inc. March 2007.

[59] Gegick, M., Williams, L. *Matching attack patterns to security vulnerabilities in software-intensive system designs.* Proceedings of the 2005 workshop on Software engineering for secure systems-building trustworthy applications. 2005.

[60] Peterson, G., Steven, J. *Defining Misuse within the Development Process.* IEEE Security & Privacy. 2006.

[61] Peterson, G. *Collaboration in a Secure Development Process Part 1.* Information Security Writers Group, Information Security Bulletin, Volume 9, June 2004. Page(s) 165 – 172.

[62] Arjan Zwikker. *Webapplicatie beveiligingsproblemen.* PricewaterhouseCoopers Advisory N.V., the Netherlands. 2007.

[63] Swiderski, F., Snyder, W. *Threat Modelling.* Microsoft Press. 2004.

[64] Meijer, E., Gough, J. *Technical Overview of the Common Language Runtime*. CiteSeer.IST. 2000.

[65] Jones, K. *Filters for Pre- and Post-Processing.* JavaPro Magazine. April 2004.

[66] Kurniawan, B. *Java for the Web with Servlets, JSP and EJB: A Developer's Guide to J2EE Solutions.* Sams Publishing. April 2002.

[67] Morries, Kief. *Writing Servlet Filters*. Internet.com.

[68] Fesler, S. *Writing Servlet 2.3 Filters.* O'Reilly OnJava.com. May 2001.

[69] Kydyraliev, M. *Bypassing servlet input validation filters (OWASP Stinger + Struts example).* Open Web Application Security Project (OWASP). August 2007.

[70] Ping, L., JianYang, L. *A Change-Oriented Conceptual Framework Of Software Configuration Management.* 2007 International Conference on Service Systems and Service Management. June 2007. Page(s) 1 – 4.

[71] Kurniawan, B., Maragioglio, N. *JavaServer Faces Programming.* McGraw-Hill Companies. December 2003.

[72] Mahmoud, Q. H. *Developing Web Applications with JavaServer Faces..* Sun Developer Network (SDN), August 2004.

[73] Hightower, R., Tabor, P. *JSF for nonbelievers: JSF conversion and validation. Use JSF's conversion and validation framework to ensure data-model integrity.* developerWorks, IBM's resource for developers. 19 April 2005.

[74] Spiegelberg, E. *Ajax Form Validation Using Spring and DWR*. Sun Microsystems, Inc. java.net. 2 August 2007.

[75] Leroy, X. *Java Bytecode Verification: Algorithms and Formalizations.* Journal of Automated Reasoning. Volume 30, Issue 3-4, 2003. Page(s) 235 – 269.

[76] EUROSEC GmbH Chiffriertechnik & Sicherheit. *Session Management in Web Applications.* 2005.

[77] Arthur, J., Azadegan, S. *Spring Framework for rapid open source J2EE Web Application Development: A case study.* Proceedings of the Sixth International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing and First ACIS International Workshop on Self-Assembling Wireless Networks (SNPD/SAWN'05). 2005.

[78] Sawer, C. *Using Spring's MVC framework for web form validation.* February 2005.

[79] Scheibelhofer, K. *Using OpenCard in Combination with the JavaCryptography Architecture for Digital Signing.* Institute for Applied Information Processing and Communications (IAIK), 2000.

[80] MySQL AB. *MySQL Developer's guide.* MySQL Press. 7 April 2006.

[81] IT Governance Institute. *Control Objectives for Information and related Technology (COBIT).* Version 4.0, 2005.

[82] ISO/IEC 17799. *Information technology – Security techniques – Code of practice for information security management.* Second Edition, 15 August 2005.

APPENDICES

APPENDIX A

Preliminary Risk Assessment Template

Business Goal

Tag	Goal	Description
G1		
G2		

Business Risk

Tag	Risk	Description	Indicators		Likelihood of Occurrence	Impact	Estimated Cost	Severity (L/M/H)
BR1			BR1.1					
			BR1.2					
			BR1.3					
BR2			BR2.1					
			BR2.2					
			BR2.3					

Technical Risk

Tag	Risk	Indicators		Likelihood of Occurrence (L/M/H)
TR1		TR1.1		
		TR1.2		
TR2		TR2.1		
		TR2.2		

Business Goal to Business Risk to Technical Risk Relationship

Business Goal Tag	Business Risk Tag	Technical Risk Tag	Severity (L/M/H)

Recommended Risk Mitigation Method

Business Risk Tag	Technical Risk Tag	Risk Mitigation Method		Level of Confidence (L/M/H)
		R1		
		R2		
		R3		
		R4		

Template for Textual Description of Abuse case

General Information					
Misuse Case:	MC-#:				
Description:					
Author:					
Date:					
Includes/Extends Other Use/Abuse Cases					
Includes:					
Extends:					
MisActor					
Insider/Outsider:					
Skill sets:					
Affected Systems (System Boundaries, External Systems, Users, Stakeholders, etc)					
●					
●					
●					
Dependencies, Conditions and Assumptions					
Network:					
Application:					
Data Storage:					
Pre-condition: *(the state of the application before the abuse case is executed)*					
			Probability of Occurrence		
			Low	**Med**	**High**
Network:	MC-#-ON1:				
	MC-#-ON2:				
Application:	MC-#-OA1:				
	MC-#-OA2:				
Data Storage:	MC-#-OD1:				
	MC-#-OD2:				
Post-condition: (the state of the application after a successful abuse execution)					
Network:					
Application:					
Data Storage:					
Detection Mechanism					
			Reliability		
			Low	**Med**	**High**
Network	MC-#-DN1:				
	MC-#-DN2:				

Application	MC-#-DA1:				
	MC-#-DA2:				
Data Storage	MC-#-DD1:				
	MC-#-DD2:				
Countermeasures					
			Requirements Tag		
Network	MC-#-CN1:				
	MC-#-CN2:				
Application	MC-#-CA1:				
	MC-#-CA2:				
Data Storage	MC-#-CD1:				
	MC-#-CD2:				

APPENDIX B

Java: An Overview

There is an intense publicity surrounding Sun Microsystems's Java technology as *the* language for programming secure web applications. Java began as a high level object oriented programming language used to develop small applications other than web applications. Over the years, Java underwent several stages of maturity and today, amongst all other web development languages, Java is perceived to be the most suitable language that is able to meet the varying requirements of today's dynamic web applications.

The Java Platform, Enterprise Edition 5 (Java EE 5) formerly known as Java 2 Platform, Enterprise Edition (J2EE) offers a multi-tiered distributed application model for the design and development of portable, robust, scalable, and secure Java enterprise web applications. It is based on well-defined components that are reusable, providing both server-side and client-side support for developing multi-tier applications [42]. J2EE primarily consists of the following important libraries JNDI, RMI, RMI/IIOP, JTA, JTS, JMS, JSP, and Servlet API while the new Java EE 5 offers additional features such as Enterprise JavaBeans (EJB) Technology 3.0, JavaServer Faces (JSF) Technology, and the latest web services APIs.

Multi-tier applications like the Java EE 5 architecture consists of three tiers; a client tier, a middle tier and a back-end tier. The client tier provides support for a variety of client types like web browsers, web pages, Java applets or standalone Java based programs running on laptops, desktops, palmtops, and cell phones. The middle tier (also known as the server tier) receives and processes client requests by executing complex business logics, connecting to other legacy or new systems, catering for session and transaction management, enforcing security, pooling resources as well as accessing database systems. This tier uses two distinct containers; web containers that manages and services JSP and servlet components; EJB containers which support the application's business logic services. Tomcat from Apache is one example of a web container. The final tier, the back-end tier, also known as the Enterprise Information System (EIS) tier, supports and manages the application's critical data and functions. This tier is accessible via Java's standard APIs.

There are mainly four different components in the Java EE 5 specification. The components are

1. Applets
2. Application Client Components
3. Servlets and JSP Components (also known as web components)
4. Enterprise Java Beans (EJB) Components

In its early days, Java received much attention for its ability to create applets. Applets are small applications written in Java that executes in the Java Virtual Machine (JVM) installed in specific web browsers. The rise of newer server side technologies enabling browser independent execution of web applications caused the declining rate of applets in web applications. These new technologies are servlets and JavaServerPages (JSP).

Servlets are modules comprising of Java code that are processed at the server-side. Each servlet for a web application is a Java class that extends or inherits the HttpServlet class. Servlets also have access to the entire family of Java APIs as well as a library of HTTP-specific calls, receiving all the benefits of the Java language, including portability, performance, reusability, and crash protection [39]. Once a servlet is compiled, it is executed by the servlet engine. A servlet can also be run by a servlet engine in a restrictive sandbox environment, which helps protect against malicious Servlets [52]. A sandbox environment in Java limits the request or access made to resources running on the application server.

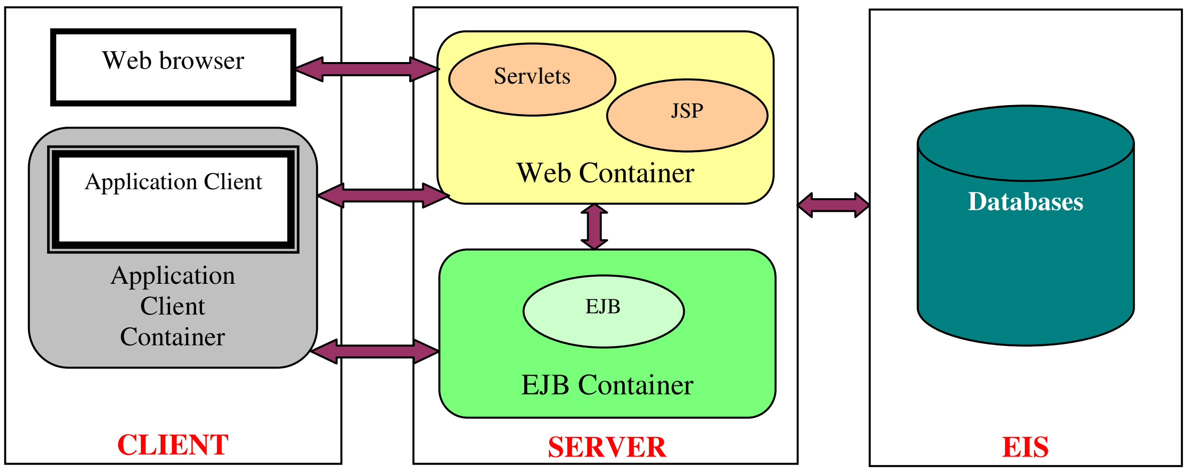


Figure 18: Multi-tier Architecture of Java EE 5

An application client component is a Java application that resides on a client machine and accesses enterprise bean components on the J2EE server [43].

JavaServer Pages (JSP) is a server-side scripting language used to create dynamic web page contents. It is an extension of the servlet technology created to support HTML and XML pages. The syntax and semantics of this language is close to that of the Java language. In fact, JSP inserts Java code in between HTML tags. A typical JSP file (with a .jsp extension) is first compiled into a Java servlet, which is handled by the servlet engine. The servlet engine then loads the servlet class (using a class loader) and executes it to create dynamic HTML contents that is displayed on the browser [38, 39]. An illustration of this process, modified from [38], is shown in Figure 19.

The main advantage of JSP in web applications is that it is platform independent. It can work with a wide variety of web servers, application servers, browsers and development tools [38, 39].

To execute a Java based web application, it is imperative that the web server is configured to run servlets and a JSP engine/container. A JSP engine/container compiles and converts a JSP page into a servlet. The compiled servlets, generally class files are stored in servlet containers residing in the Java Virtual Machine (JVM) and are usually used for back-end processing. The JSP page is loaded only once in the container before it remains permanent. When the JSP page is modified, the server has to be restarted or the particular application should be reloaded for the changes to take effect.

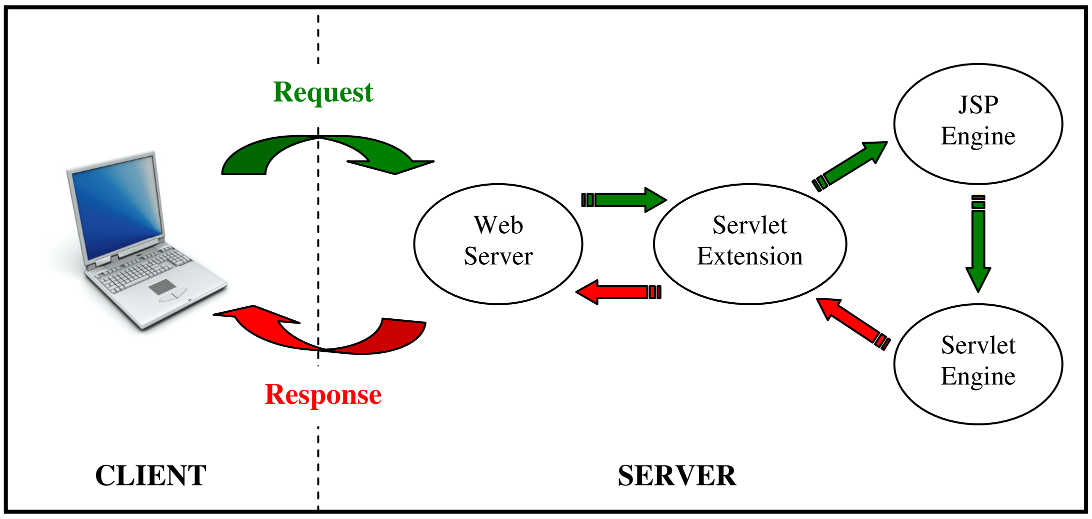


Figure 19: JSP and Servlet execution in Java based web applications (a)

Complimentary to Figure 19 is Figure 20:

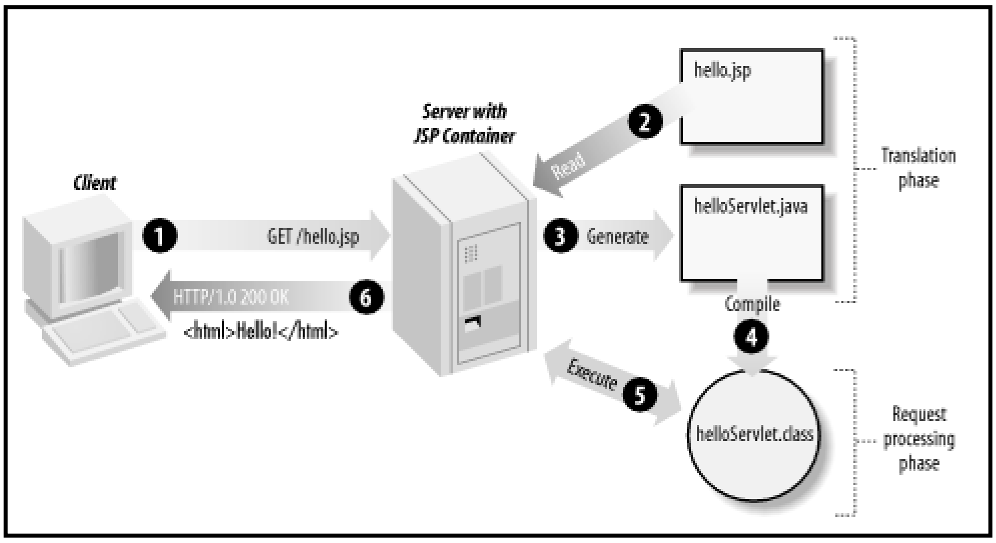


Figure 20: JSP and Servlet execution in Java based web applications (b)

More clearly, a client requests for a *hello.jsp* page. The JSP engine/container converts the JSP code into a servlet (*helloServlet.java*) and compiles the servlet (*helloServlet.class*). Then, the JSP engine/container loads that servlet into the servlet engine, which runs it. For subsequent requests of *hello.jsp* from the same client, the JSP engine/container runs the servlet that corresponds to the JSP.

For a servlet and JSP engine/container to work, it must have access to Java's *Software Development Kit*, or *SDK* which is a part of the *Java 2 Platform, Standard Edition (J2SE).* The *SDK* comprises of a Java Development Kit (JDK) which contains Java classes and a Java Runtime Environment (JRE) that altogether supports the development of Java Web Services. JRE provides for the libraries, the Java Virtual Machine (JVM) and other related components to run applets and applications developed in Java. The JDK, on the other hand, is a superset of the JRE, containing everything that is in the JRE as well as additional tools such as compilers and debuggers necessary for developing applets and applications [53].

Yet another important component, Enterprise JavaBeans (EJBs) and EJB server/container, which enables rapid and simplified development of distributed, transactional, secure and portable applications, may also be used for developing web applications using Java. EJB is the Java technology aiming to be the preferred choice for web applications using Java language An EJB consists of an EJB component, an EJB container and an EJB object.

An *EJB component* is a Java class that implements business logic. An *EJB component* executes within an *EJB container* which is contained inside an EJB server. Any server that can host an *EJB container* and provide it with the necessary services can be an EJB server. An *EJB container* provides services such as transaction and resource management, versioning, scalability, mobility, persistence, and security to the EJB components it contains. Lastly, the *EJB object* runs on the client, providing the user interface logic which will remotely execute EJB component's methods [40].

APPENDIX C

ASP.NET: An Overview

Having realized the growth in distributed computing environments and the importance of integration between these computing platforms and applications, Microsoft reinvented their former ASP (Active Server Pages) technology to today's popular ASP.NET technology. Predominantly, ASP was Microsoft's original technology in developing dynamic web sites. However, the increasing interactive functionality in web applications led to the inception of the .NET Framework which can be used to create both computer and web based applications. The part of .NET dedicated to web applications is called the ASP.NET.

The ASP.NET is not an extension of the classic ASP but is an entirely new technology, encompassing the features of classic ASP, for developing web based applications and services. It is built on the Microsoft .NET Framework and is executed within Microsoft's Internet Information Services (IIS).

Different from ASP, applications written in ASP.NET are compiled instead of interpreted at the server during processing. The interpreted code model of ASP limited the performance of applications written in ASP. Applications written in pure ASP.NET language are text based files with the .aspx extension. However, ASP.NET applications can also consist of HTML files, Cascading Style Sheet (CSS) files, and even script oriented files like JavaScript (.js), VBScript (.vbs), Perl (.pl), etc.

The technologies that fall under the umbrella of the .NET security framework particularly for web applications include [49]:

IIS	Microsoft's web server application that runs only on computers using the Windows operating system.
ASP.NET	Web based programming technology for Microsoft based web applications.
Enterprise Services (COM+ Services)	Runtime environment for objects, providing 4 major services like automatic transactions, queued components, loosely coupled events, and role-based security, that helps in the building of scalable distributed systems [54].
Web Services	Provides a standardized means to integrate with various distributed web based applications/services using technologies like XML, SOAP, WSDL and the like regardless of operating system or programming language.
.NET Remoting	A technology that enables the communication between applications distributed across multiple domains, processes, and boundaries.
SQL Server and ADO.NET	The SQL server is Microsoft's Relational Database Management System (RDBMS). ADO.NET provides data access services to the SQL Server. It is designed specifically for distributed web applications.

Table 9: .NET security framework technologies

As in other interactive applications, web applications developed in .NET consists of three logical services (alike to the tier architecture) namely:

1. User Services (Presentation Tier)
   Provides users an interface to interact with the core business logic of the system.

2. Business Services (Business Tier)
   Provides for the core functionality of the system.

3. Data Services (Data Tier)
   Responsible for providing access to retrieve, store, and update data hosted by the system or to other backend systems.

Each service will communicate only to the services adjacent to it although these services may either be physically located on the same computer or spread across multiple computers.

The .NET Framework has two core components: the Common Language Runtime (CLR) and the Framework Class Libraries (FCL)

- **The Common Language Runtime (CLR)**
  A language neutral development and execution environment, similar to the JVM (Java Virtual Machine) that supports a wide range of programming languages and computing platforms [62]. With the CLR, applications can be coded using any of the compatible languages (C#, Visual Basic, Jscript, J#) whose compiler conforms to the common language specification (CLS). It is designed in a way that allows one language to be naturally extended to code written in another programming language.

  When an application is launched for the first time, a .NET compiler converts the source code into the Common Intermediate Language (CIL) code. The CIL is a CPU and platform independent set of instructions that are converted at runtime by the CLR's JIT (Just In Time) compiler into an assembly (portable executable file) which is native to the operating system. JIT compilation provides environment-specific optimization, runtime type safety, and assembly verification [16]. The performance of the application also increases as all later executions no longer require the CIL to native code compilation.

  Managed code is the term applied to any software running on the .NET Framework including but not limited to C#, Java, C++, Visual Basic, PERL, etc. It is packaged as a binary file consisting of CIL and metadata. The JIT compiles the managed code into machine language. While the application's logic is described by the CIL, the metadata describes the parts of the code related to class definitions, method definitions, parameter types, return types, and just about every other aspect of the code, other than the code itself.

  Some prevailing advantages of managed code in CLR are

  i. *Memory Management*: well managed and maintained heap used for memory allocations.
  ii. *Security*: un-trusted code (binaries downloaded and executed from malicious websites) can be detected during JIT compilation. If, however, the un-trusted code is able to bypass JIT, the CLR is still able to react by raising a security exception.
  iii. *Thread Management*: similar with memory management except that instead of removing objects that are never used, unused threads are returned to a queue until it is needed again.
  iv. *Type Safety*: At runtime, the CLR checks if all typecasting operations are valid.

- **The Framework Class Libraries (FCL)**
  A consistent pre-packaged functionality holding a huge collection of reusable and extendible object-oriented libraries . The framework consists of many classes that perform a variety of tasks ranging from file reads and writes, GUI programming, advanced cryptography, etc.

The components that make up an ASP.NET web application are as depicted in Figure 21 below.

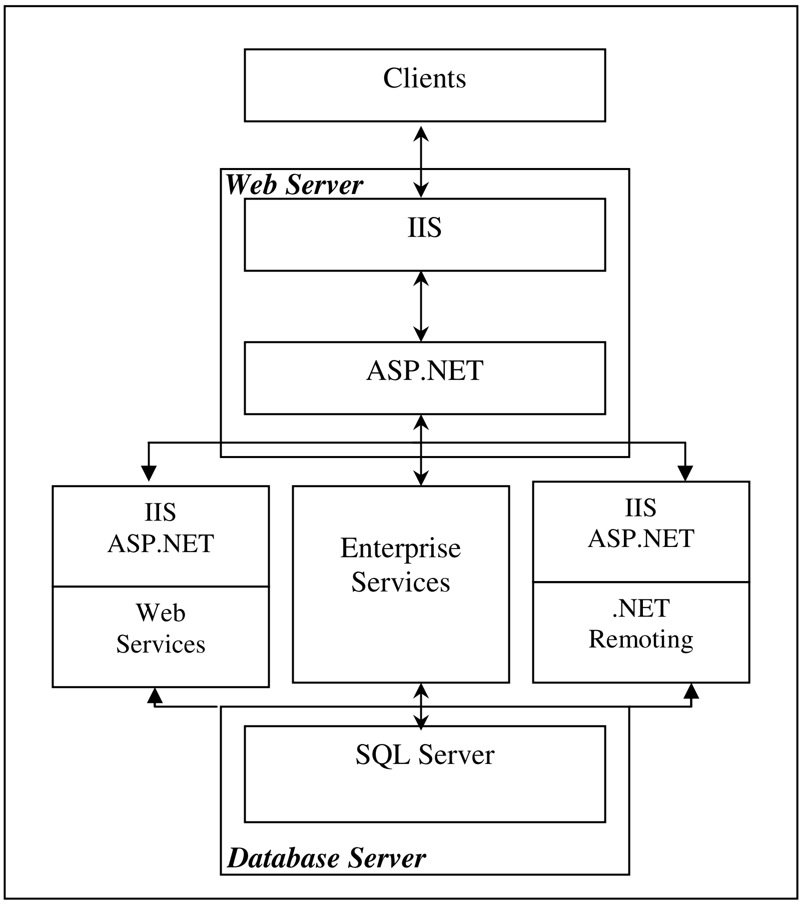


Figure 21: Components of an ASP.NET application

Clients, consisting of web browsers make a web page request to the web server (IIS). IIS then checks if the request if for a HTML or ASP.NET page. The ISAPI.dll which resides inside the IIS examines the file extension of the requested file and determines which ISAPI extension should handle the request. If the request is for a HTML page, the IIS retrives the file from the operating system and returns it to the client (web browser). On the other hand, if the request is for an ASP.NET page, IIS deciphers and optionally authenticates the request before passing it over to the ASP.NET runtime which will then process the application and return a response to the client.

When a browser requests for an ASP.NET file, the ASP.NET application framework is created for the first time. Upon the first request, the framework constructs a pool of HttpApplication instances which are automatically assigned to process incoming HTTP requests received by the application. At the time the first request is made, the Application_Start event is raised. The assigned HttpApplication instance manages the entire lifetime of a HTTP request and is even reused after a request has been completed. For performance optimization, HttpApplication instances are reused for multiple requests. The pool of HttpApplication instances continues to process incoming requests, until the last instance exits and the Application_End event is raised.

ASP.NET comes with a myriad of features. Below describes in brief some of its main features

1. *Easy Programming Model with Enhanced Reliability*
   An easy programming model provided by ASP.NET Server controls which are tags that are understood by the server. There are three kinds of server controls:
   i. HTML Server Controls – Traditional HTML tags
   ii. Web Server Controls – New ASP.NET tags
   iii. Validation Server Control – Exclusively used for input validation
2. *Flexible language option and support*

- supports more than 25 .NET languages.
- built in support for Visual Basic, C#, Jscript, C++
- uses the new ADO.NET

3. *Comprehensive Tool Support*
   - especially when using Visual Studio .NET
   - integrated support for debugging and deploying ASP.NET web applications
   - even assist organization to plan, analyze, build, test and coordinate the development of applications in ASP.NET as it includes UML class modelling, Database modelling, testing tools for functional, performance and scalability, etc.

4. *Rich Class Framework*
   Offers various built-in classes that encapsulate rich functionality like XML, data access, file upload, performance monitoring and logging and many others

5. *Compiled execution*
   Automatically detects changes, dynamically compiles files if needed and stores the compiled results to reuse for subsequent requests

6. *Rich Output Caching*
   Output Caching is configurable; can choose to cache individual regions or an entire page
   If output caching is enabled, ASP.NET executes the page just once and saves the result in memory in addition to sending it to the user. When other user requests the same page, ASP.NET serves the cached result from memory without re-executing the page. Eliminates the need to query the database on every request.

7. *Memory leak, deadlock and crash protection*
   ASP.NET automatically detects and recovers from errors like deadlocks and memory leaks to ensure application is always available.

8. *XML based components for building web services*
   No extensive knowledge of networking, XML/SOAP is required to convert any class into an XML Web Service.

APPENDIX D

Reference Card: Coding Policies and Guidelines for Web Application Security

Coding Policies and Guidelines for Web Application Security by Sabrina Samuel

Buffer Overflows

- Use compiled/ interpreted & strongly typed languages (Java,ASP.NET)
- Prevent unauthorized access/tampering on application memory
- Only filtered & validated data are sent to memory

Java (via JVM provides)
- Type safety
- Class loader & byte-code verifier
- Bound checking
- Garbage collection system

ASP.NET (via CLR provides)
- Type safety
- Code verification
- Automatic memory management
- Garbage collection system

Mandatory Security Requirements

– Check length of input before accepting into memory
– Reserve adequate amount of data/virtual address space for stacks/arrays/heaps
– Check code for use of unsafe functions/keywords
  o NULL input pointers?, off-by-one errors?, truncation errors?
– Validate URL for commands/symbols/extensions
– Review use of all mathematical and logic operators for off-by-one errors
– Free allocated/reserved memory space after use
  o Remove/delete all associated contents/resources

Configuration Management

- Create/maintain separate configuration guideline for all servers
- Store configuration information in secure locations
- Configuration information is only accessible to authorized users
- Update inventory of applications/files/databases on the servers
- Perform regular vulnerability scanning
- Provide periodic status reports on overall security status

Java

SVNKit	to access/modify Subversion repository

ASP.NET

Microsoft Management Console (MMC) snap-in	GUI for administrating the Web.config & Machine.config files
Web.config	Sets application configuration
Machine.config	Sets server configuration

Mandatory Security Requirements

– Use unique naming convention for all directories/files/databases
– Block URL's that include ".." & "\" characters
– Create \include directory outside the web document root
– Place sensitive data files outside /public_html & /cgi-bin
– Don't store sensitive & public data on same directory/machine
– Remove/relocate unknown/default/backups/scripts/unused files
– Set server to password-protect/IP restrict resources
– Turn off directory indexing/directory browsing in web server
– Do not accept server defaults
– Limit executable files to specific directories & disable download
– Incorporate explicit version control system

General Rules and Policies

– Use SSL/TLS for all transmissions carrying protected data
– Set the web server to NOT allow automatic switching between the HTTP and HTTPS protocol

Java

– Define HTTPS connections in Apache mod_ssl module
- HTTPConnection con = new HTTPConnection("https" , www.webAddress.net, -1);

ASP.NET

– Define HTTPS connections in the ASP.NET Web.config file, between the <secureWebPages> and <\secureWebPages> tag

It is important that the technical documentation from the architecture and design stage is flexible enough to incorporate changes to an existing web application.

Input Validation

- Identify all sources of input to the application (client browsers, database, external applications, files, etc.)
- Use client-side & server-side validation or alternative Ajax like techniques for all user input
- Ensure user input and data from client are received in right type, length, format, range.

Java

javax.servlet.Filter Package	Intercepts incoming/ outgoing requests/responses to view, extract, modify and/or process data
JavaServerFaces (JSF) Framework (*javax.faces.validator)*	Component based API for building user interfaces. Offers four forms of validation methods. Can incorporate Ajax features.
Struts Framework	Has a Validator framework that performs validations on form data. Can incorporate Ajax features.
Spring & Direct Web Remoting (DWR) Framework	Features a *Validator* and *Data Binder* interface that make up the data validation package. DWR enables Ajax functionalities.

ASP.NET

Validation Web controls	Provides validation checks for inputs received from web forms

Mandatory Security Requirements

– Determine required/optional input fields
– Check for min/max length, type(numeric, alpha, alphanumeric), format/syntax (email, credit card, etc.) and range
– Perform cross field validation for certain input
– Trim white spaces at beginning & end of input
– Runs of white space is replaced with a single space
– Encode white spaces to its HTML equivalent (%20)
– Refuse NULL characters (%00)
– If using hidden fields/cookies
  o Encrypt/hash information contained within it
– Do not use HTTP Headers to make any security decisions
– Do not allow application to auto-correct wrongly entered input
– Avoid/minimize the use of JavaScript
  o Ensure all reference to DOM objects are inspected for its use
– For restrictive controls like checkboxes, radio buttons, drop down lists and hidden fields, use index/label to match the corresponding name/value pair on the server

Form Submissions

– All Form submissions should use the POST method
– Prevent forms/transactions from being submitted multiple times
  o Generate unique, random ID linked to the form/transaction
  o Session timeouts/refresh should invalidate the request

Password Controls

– Define minimum (6-8 chars) and maximum length
– Must be a combination of upper/lower case letters, numbers, selected special characters. No use of dictionary words
– Set password expiry dates; Forbid re-use of expired passwords
– Implement account lockout policy
  o Wrong passwords (3-5 times); disable account & kill session
  o Implement delay before re-accessing lock accounts
– Hash/Encrypt Passwords before storing
– Retain old hashed/encrypted password in separate location
– Use CAPTCHA to prevent use of unauthorized scripts/programs
– Forgotten/Change/Unlock Passwords
  o Use secret question(s)&answer(s)/pass phrases
  o For non-critical applications, send newly generated password + timestamp to email address
  o Prevent changing of passwords often

Secure Storage

- Store sensitive information in encrypted formats
- Protect resources from unauthorized access/disclosure/modification/destruction

Java

java.security (JCA)	basic cryptographic services & algorithms including message digests, ciphers, key generators and key factories
javax.crypto (JCA, JCE)	encryption support for symmetric, asymmetric, block, & stream ciphers for standard well-known algorithms
Java keystore	stores keys and trusted certificate in a protected database

ASP.NET

System.Security.Cryptography	Symmetric and asymmetric cryptography, hashes, digital signatures, signed/enveloped messages and random number generation
Configuration settings in Web.config	Used to encrypt configuration information
Data Protection API (DPAPI)	Encrypts data using current user account or computer information. Also protects the key use for encryption.

Mandatory Security Requirements

– Browser cache's must be automatically deleted/cleared when user logs out/browser window closes
– Stick to existing standard cryptographic algorithms
– Reminder for encryption techniques:
  o Use strong random key generators; Ensure key size is sufficient
  o Store keys in restricted location with restricted access
  o Expire keys regularly
– Take into account the Certificate Revocation List (CLR) in case of PKI

Coding Policies and Guidelines for Web Application Security by Sabrina Samuel

Injection Flaws

- Filter & validate all input received by the application
- Include data conversions and regular expressions

Java

java.util.regex	Specifies acceptable textual patterns

ASP.NET

System. Text.RegularExpressions	Specifies acceptable textual patterns

Mandatory Security Requirements

– Regular expressions to match authorized/unauthorized contents
– Examine all calls made to external systems/interpreters
– Prevent execution of operating system/SHELL commands
– Handle input data literally and not as executable content
– Application must run with right privileges when executing functions across boundaries
– Avoid accepting file names as input
 ○ Use indexes when trying to retrieve files

Access Control

- Describe & document a detailed access control policy
- Review code that implements access controls
- Plan & perform rigorous access control testing
- Limit access to protected contents or data

Java

Java Authentication and Authorization Services (JAAS

Core Classes	Policy, AuthPermission, PrivateCredentialPermission

ASP.NET

File Authorization	Uses ACLs of the resources
URL Authorization	Associates users/roles to URL

Mandatory Security Requirements

– Identify a suitable access control model (i.e. RBAC/Lattice model)
– Use strong randomly generated session ID/token
– Separate access control code from application code
– Perform authorization checks on every protected page/URL
– Restrict access to administrative pages/sections
– Prevent use of web interface for administrator access
 ○ Otherwise ensure use of VPN/SSL technology
– Application should only execute with the right privileges
– Set expiry property for protected contents/resources
– Change/verify defaults if use/access external code/applications

Application Denial of Service

- Thorough Input Validation
- Properly handle errors and exceptions
- Avoid unnecessary accesses to databases/sensitive resources

No specific APIs/frameworks in Java & .NET to prevent application DoS attacks.

Mandatory Security Requirements

– Connections to protected contents/data must have timeouts
– Limit resources allocated to a user at any one point
– Avoid high CPU consuming operations
– Limit the load a user can put on the application
– Handle only one request per user at a time
– Set timeouts in all user sessions
– Analyze/monitor size of log files

Cross Site Scripting

- Filter & validate all user input
- Encode all special characters into its ASCII/UNICODE/ISO format
- Encode all output data before displaying on the browser

Java

java.net.URLEncoder	Performs both URL encoding and decoding
StringEscapeUtils	Escapes/unescapes string literals for Java, JavaScript, HTML, XML, and SQL

ASP.NET

HttpUtility.HtmlEncode	**Encodes all special characters**
HttpUtility.UrlEncode	Performs both URL encoding and decoding

Mandatory Security Requirements

– Set the character set and language locale of each web page
– Use White List to define acceptable HTML tags or special characters
– Check the attribute & value elements of permitted HTML tags
– Disable HTTP TRACE & HTTP TRACK methods
– Make sure application does not repeat any encoding/decoding process
– Perform security checks on decoded data
– Restrict application to use only one canonical format
–Decode encoded characters before storing into the database

Error Handling and Logging

- Deal with all failure & exceptional conditions in a secure manner
- Proper formulation of error/exception messages for end users
- Periodically review log files for misuse or leaked information

Java

java.lang.Throwable	Throws all errors/exceptions
java.lang.Exception	For abnormal conditions to be caught
java.lang.Error	For serious problems an application can't handle
java.util.logging	Capture information on security failures, configuration errors, and other information

ASP.NET

Error Handling	Page_Error: traps errors at page level Application_Error: trap errors at application level customErrors: for customized error messages
Exception Handling	SystemException class for run-time errors ApplicationException for non-fatal errors

Mandatory Security Requirements

– Always use try/catch/finally blocks around code
– Avoid writing try/catch/finally blocks with empty catch block
– Use catch/finally blocks to cleanup or release resources
– Formulate concise error messages for all expected/unexpected events
– Remove sensitive information/comment embedded within the code
– Ensure application has a safe mode to recover from unexpected events
– Configure servers to NOT show default/verbose error messages
– Redirect errors to a generic page
– Do not reveal detailed error information (i.e. stack trace)
– Default error/debugging information only for server admin. group
– Application should provide generic messages with tracking numbers
– Create application-level logs
– Store log files in secure locations only accessible to the admin. group
– Secure log files using access control lists
– Always maintain a backup copy or archive log files
– Log all output, return code and errors generated by the application
– Session ID/token is used to link users to malicious attempts
– Ensure that logs are not overwritten/tampered by local/remote users
– Periodic review of log files to detect any fault/suspicious activity

SQL Injection

- Filter & validate user input for special characters and reserved words in SQL
- Use stored procedures/prepared statements with parameterized SQL
- Enforce Least Privilege policy using access control credentials

Java

java.sql.PreparedStatement java.sql.statement	Implements prepared statements
CallableStatement	Enables JDBC to invoke stored procedures

ASP.NET

SqlCommand & SqlParameter	Implements prepared statements/stored procedures

Mandatory Security Requirements

– Do not concatenate SQL strings that include user input
– Use parameterized SQL to query the database
– Execute stored procedures using a safe interface
– Assign permissions (write/execute) on stored procedures
– Avoid disclosing detailed database error information
– Do not embed database names in application code
– Validate the number of rows returned from a query

Authentication and Session Management

- Choose appropriate form of authentication per the application's risk
- Re-authenticate users for high value transactions/protected information
- Use strong session ID & protect them throughout their lifecycle

Java

Java Authentication and Authorization Services (JAAS)

Common Classes	Subject, Principals, Credential
Core Classes	LoginContext, LoginModule, Configuration, CallbackHandler

ASP.NET

IIS Authentication	Basic, Digest, Integrated Windows, Client Certificate, Anonymous
Authentication Providers	Form Authentication, Windows Authentication

Mandatory Security Requirements

– Adopt strong authentication for highly sensitive information
– Encrypt/hash sensitive authentication/session data
– Separate HTTP & HTTPS requests
– Configure server to require authentication at directory levels
– Authenticate components (internal/external code/application)
– Disable/lock accounts that have not been accessed for sometime
– Implement delay between submitted credential & success/failure
– Inform users of last login date/time & any failure attempts since then
– Avoid use of "Remember Me", "Sign me automatically", etc. options
– All authentication pages must be marked with the "no-cache" tag
– Use/create random/unique algorithm for session ID/tokens
– Ensure users get new session ID/tokens with each visit/revisit to site
– Allow only one session per user at a time. Block all other access
– Session ID/token must always refer to information stored on server
– Do not store critical session info. in cookies/URLs/hidden fields
– Restrict the sequence in which users can view pages
– Avoid use of permanent cookies, don't rely on cookie's expiry date
– Restrict cookies to specific sites/sections, encrypt cookies
– Expire/Destroy/Lock session ID/tokens when browser/application is closed or user remains idle
– Prevent users from reactivating expired session ID/token
– Expire URLs at points to limit access without re-authentication